

CprE 281: Digital Logic

Instructor: Alexander Stoytchev

<http://www.ece.iastate.edu/~alexs/classes/>

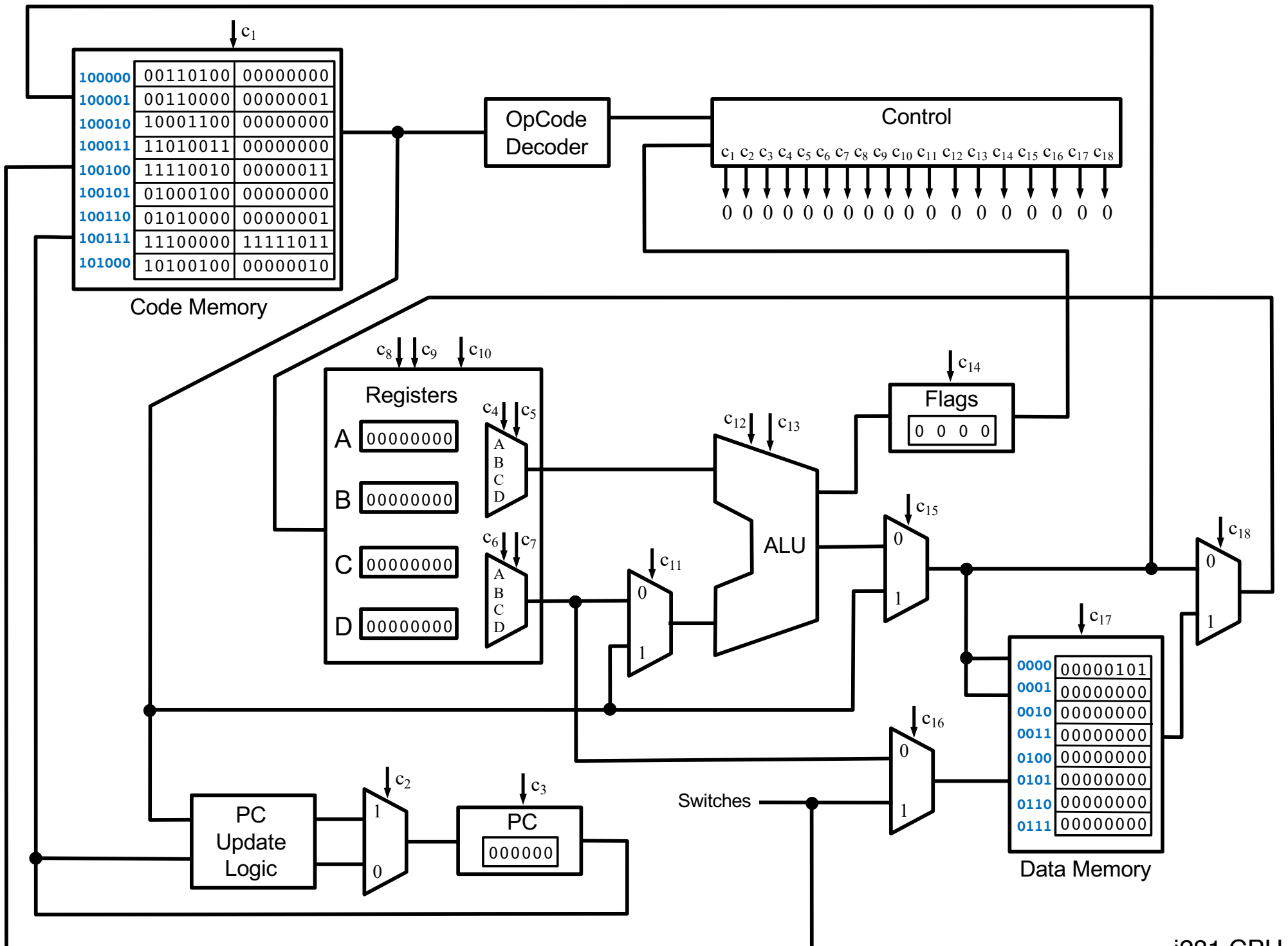
Assembly Examples

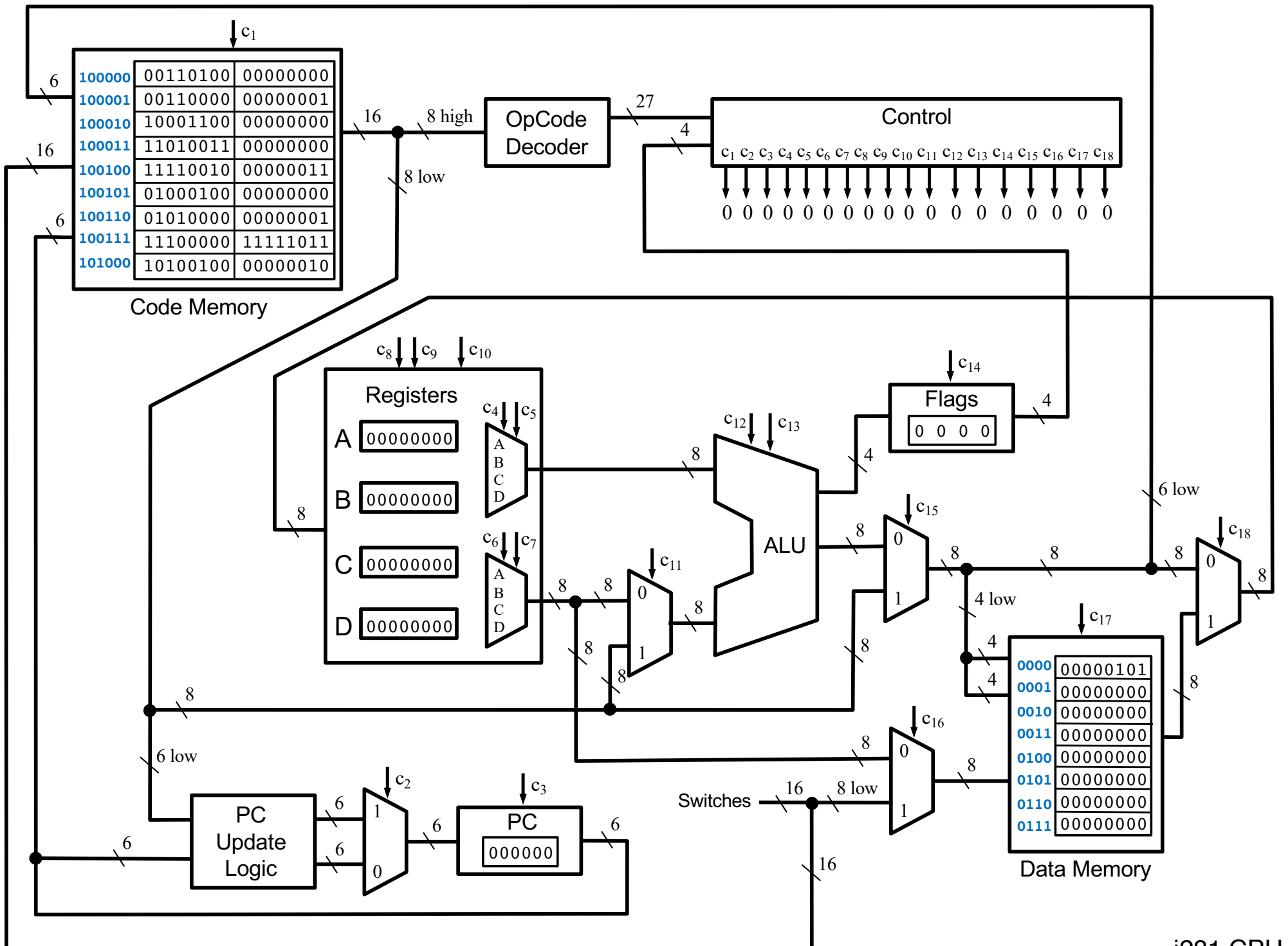
CprE 281: Digital Logic
Iowa State University, Ames, IA
Copyright © Alexander Stoytchev

Assembly Examples

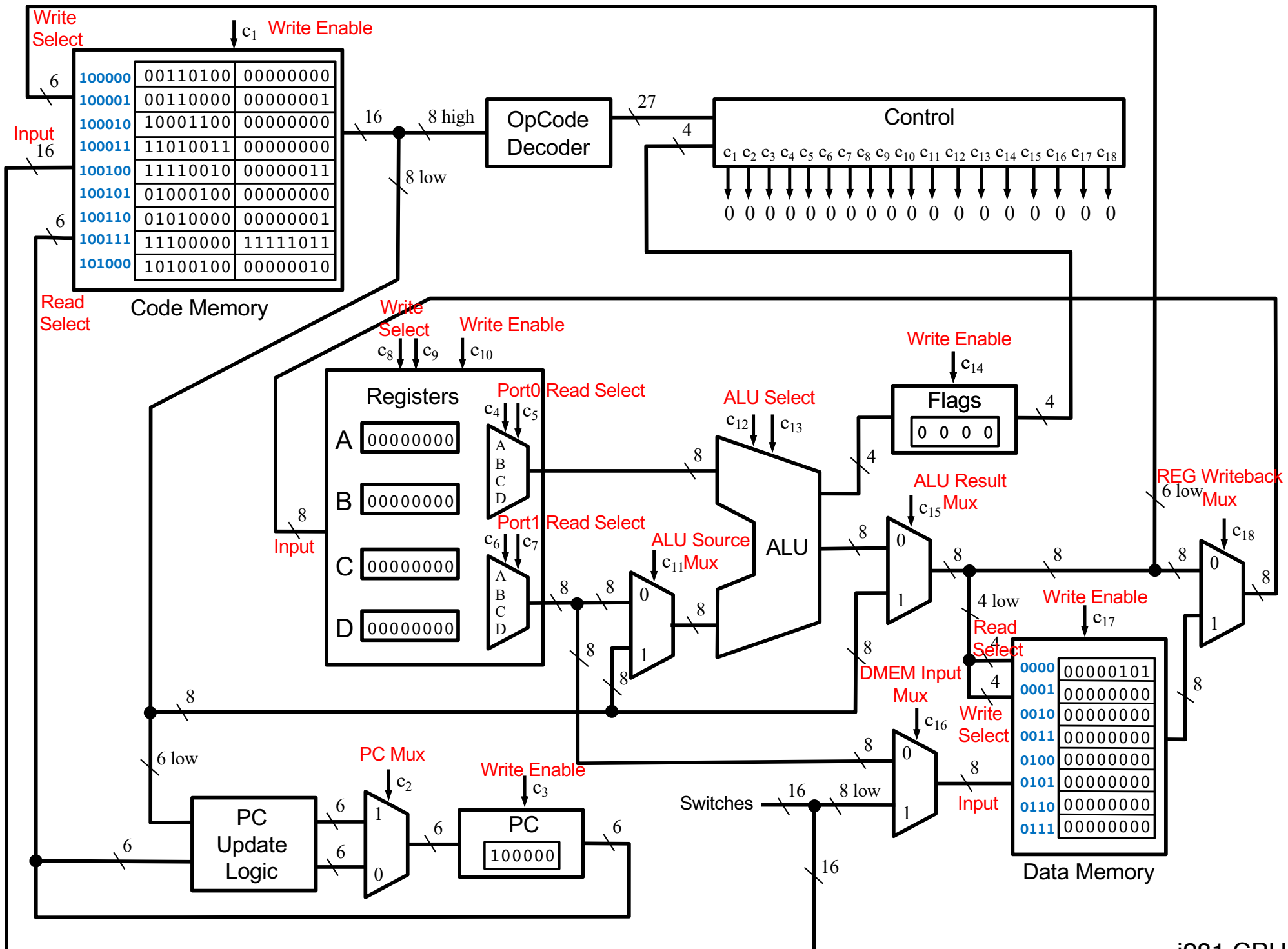
(for the i281 CPU)

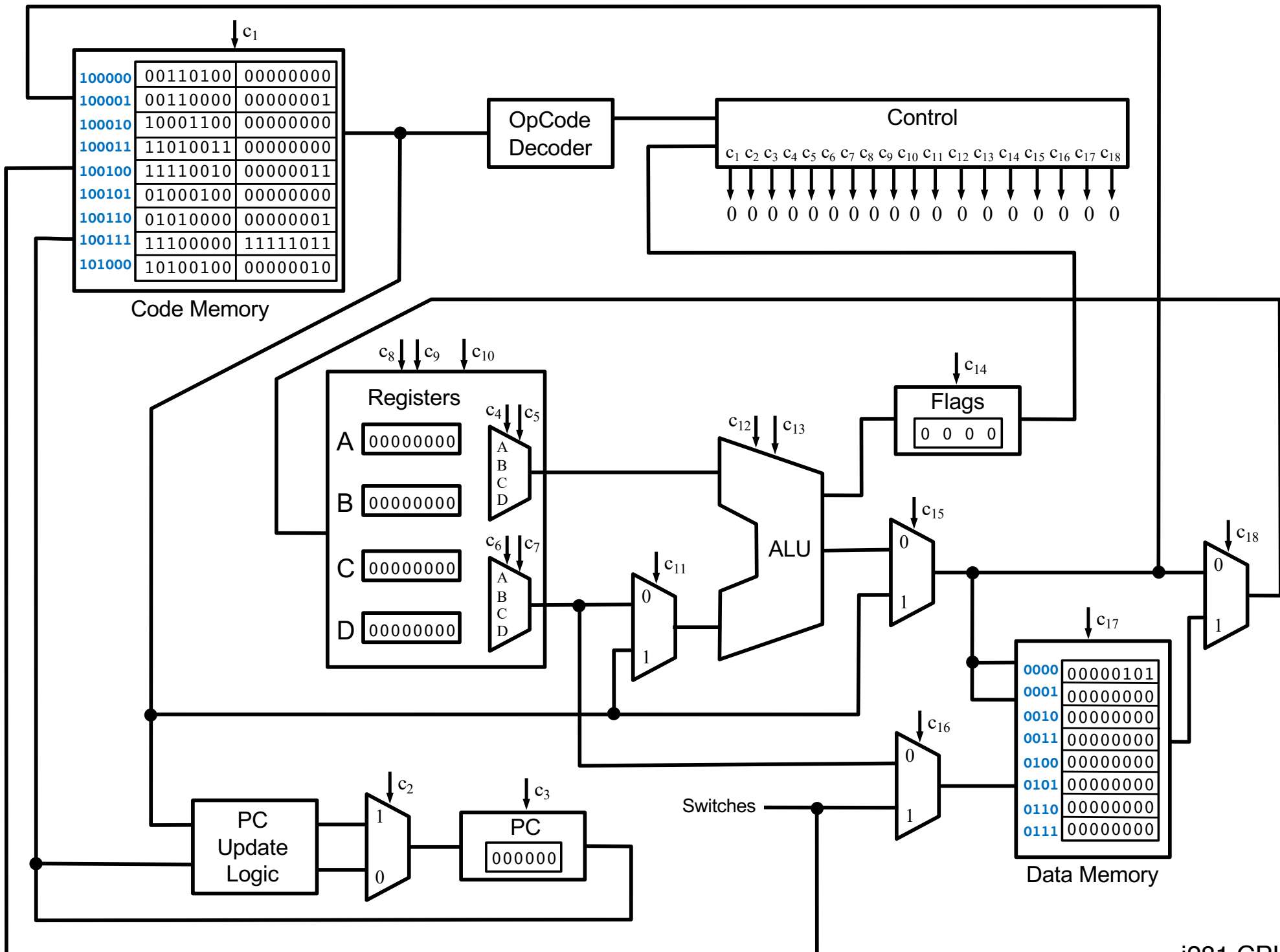
CprE 281: Digital Logic
Iowa State University, Ames, IA
Copyright © Alexander Stoytchev





i281 CPU





The Assembly Language Instructions

The i281 Assembly Instructions

NOOP	NO OPERATION
INPUTC	INPUT into Code memory
INPUTCF	INPUT into Code memory with offset
INPUTD	INPUT into Data memory
INPUTDF	INPUT into Data memory with offset
MOVE	MOVE the contents of one register into another
LOADI	LOAD Immediate value
LOADP	LOAD Pointer address
ADD	ADD two registers
ADDI	ADD an Immediate value to a register
SUB	SUBtract two registers
SUBI	SUBtract an Immediate value from a register
LOAD	LOAD from a data memory address into a register
LOADF	LOAD with an offset specified by another register
STORE	STORE a register into a data memory address
STOREF	STORE with an offset specified by another register
SHIFTL	SHIFT Left all bits in a register
SHIFTR	SHIFT Right all bits in a register
CMP	COMPARE the values in two registers
JUMP	JUMP unconditionally to a specified address
BRE	BRanch if Equal
BRZ	BRanch if Zero
BRNE	BRanch if Not Equal
BRNZ	BRanch if Not Zero
BRG	BRanch if Greater
BRGE	BRanch if Greater than or Equal

The i281 Assembly Instructions

NOOP	NO OPERATION
INPUTC	INPUT into Code memory
INPUTCF	INPUT into Code memory with offset
INPUTD	INPUT into Data memory
INPUTDF	INPUT into Data memory with offset
MOVE	MOVE the contents of one register into another
LOADI	LOAD Immediate value
LOADP	LOAD Pointer address
ADD	ADD two registers
ADDI	ADD an Immediate value to a register
SUB	SUBtract two registers
SUBI	SUBtract an Immediate value from a register
LOAD	LOAD from a data memory address into a register
LOADF	LOAD with an offset specified by another register
STORE	STORE a register into a data memory address
STOREF	STORE with an offset specified by another register
SHIFTL	SHIFT Left all bits in a register
SHIFTR	SHIFT Right all bits in a register
CMP	COMPARE the values in two registers
JUMP	JUMP unconditionally to a specified address
BRE	BRanch if Equal
BRZ	BRanch if Zero
BRNE	BRanch if Not Equal
BRNZ	BRanch if Not Zero
BRG	BRanch if Greater
BRGE	BRanch if Greater than or Equal

The OPCODEs

(Mapped to Machine Language)

The OPCODEs

NOOP

0	0	0	0	d	d	d	d	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

INPUTC

0	0	0	1	d	d	0	0	C	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

INPUTCF

0	0	0	1	R	X	0	1	C	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

INPUTD

0	0	0	1	d	d	1	0	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

INPUTDF

0	0	0	1	R	X	1	1	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

MOVE

0	0	1	0	R	X	R	Y	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

LOADI/LOADP

0	0	1	1	R	X	d	d	I	M	M	E	D	V	A	L
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

The OPCODEs

ADD

0	1	0	0	R	X	R	Y	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ADDI

0	1	0	1	R	X	d	d	I	M	M	E	D	V	A	L
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

SUB

0	1	1	0	R	X	R	Y	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

SUBI

0	1	1	1	R	X	d	d	I	M	M	E	D	V	A	L
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

LOAD

1	0	0	0	R	X	d	d	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

LOADF

1	0	0	1	R	X	R	Y	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

STORE

1	0	1	0	R	X	d	d	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

STOREF

1	0	1	1	R	X	R	Y	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

The OPCODEs

SHIFTL

1	1	0	0	R	X	d	0	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

SHIFTR

1	1	0	0	R	X	d	1	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

CMP

1	1	0	1	R	X	R	Y	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

JUMP

1	1	1	0	d	d	d	d	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

BRE/BRZ

1	1	1	1	d	d	0	0	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

BRNE/BRNZ

1	1	1	1	d	d	0	1	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

BRG

1	1	1	1	d	d	1	0	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

BRGE

1	1	1	1	d	d	1	1	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Sample Assembly Programs for the i281 CPU

Arithmetic

C Version

```
// Arithmetic1  
//  
// C version
```

```
int main()  
{  
    int x=2;  
    int z;  
  
    z = x+3;  
  
    // printf("%d\n", z);  
}
```

Assembly Version

```
; Assembly version
```

```
.data
```

```
x          BYTE      2
z          BYTE      ?
```

```
.code
```

```
    LOAD  A, [x]
    MOVE  C, A      ; z=x;
    ADDI  C, 3      ; z+=3;
    STORE [z], C    ; update the memory for z
```

```
; Register allocation
```

```
;
```

```
; A: x
```

```
; B: <not used>
```

```
; C: z
```

```
; D: <not used>
```

C vs. Assembly

```
// Arithmetic1
//
// C version

int main()
{
    int x=2;
    int z;

    z = x+3;

    // printf("%d\n", z);
}
```

```
; Assembly version

.data
x          BYTE    2
z          BYTE    ?

.code
LOAD      A, [x]
MOVE      C, A          ; z=x;
ADDI     C, 3          ; z+=3;
STORE    [z], C        ; update the
                       ; memory for z
```

Arithmetic 2

C Version

```
// Arithmetic 2
//
// C version

int main()
{
    int x=2;
    int y=3;
    int z;

    z = x+y;

    // printf("%d\n", z);
}
```

Assembly Version

```
; Arithmetic 2  
; Assembly version
```

```
.data
```

```
x          BYTE      2  
y          BYTE      3  
z          BYTE      ?
```

```
.code
```

```
    LOAD  A, [x]  
    LOAD  B, [y]  
    MOVE  C, A      ; z=x;  
    ADD   C, B      ; z+=y;  
    STORE [z], C
```

```
; Register allocation
```

```
;
```

```
; A: x
```

```
; B: y
```

```
; C: z
```

```
; D: <not used>
```

C vs. Assembly

```
// Arithmetic 2
//
// C version

int main()
{
    int x=2;
    int y=3;
    int z;

    z = x+y;

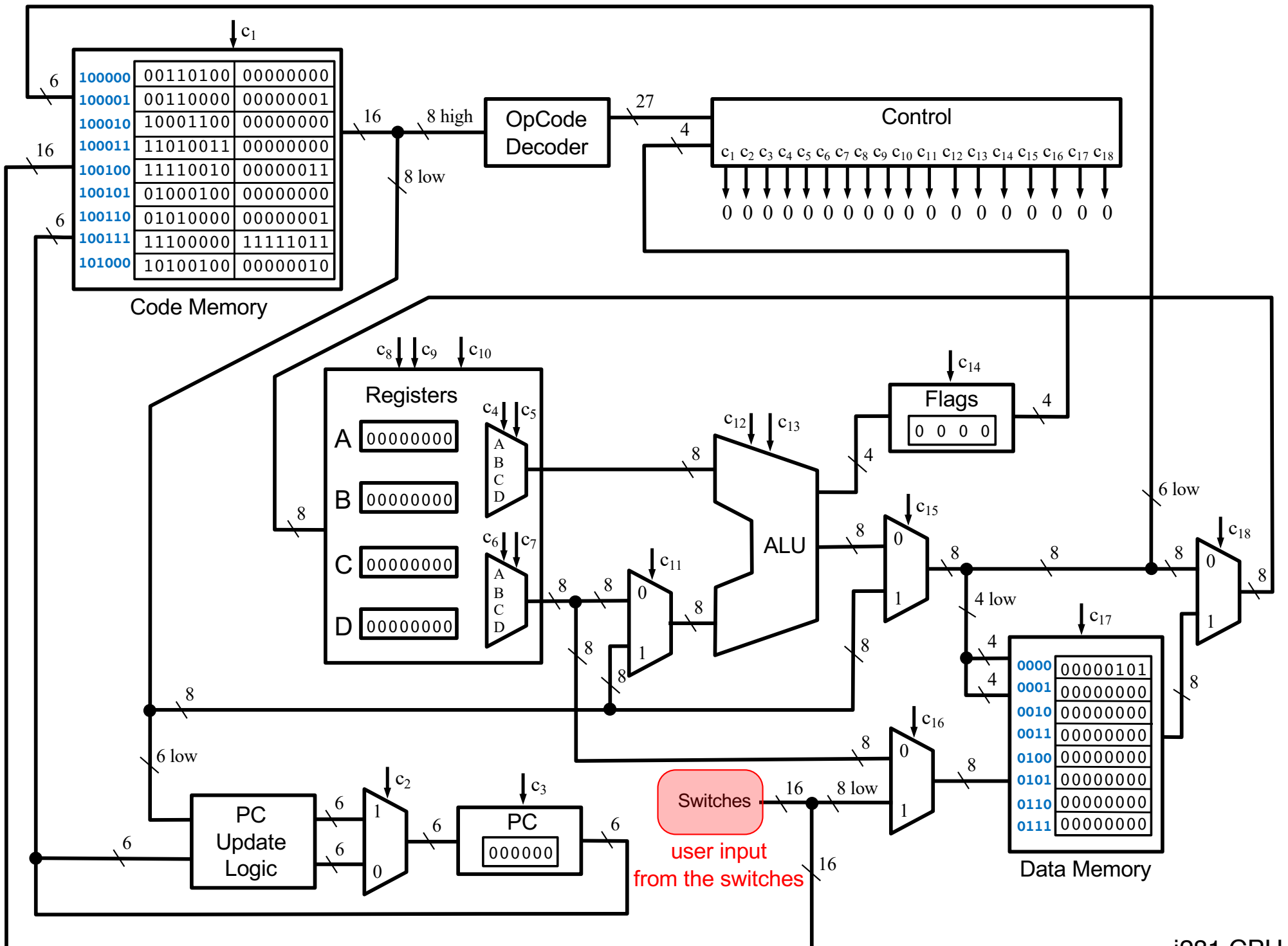
    // printf("%d\n", z);
}
```

```
; Arithmetic 2
; Assembly version

.data
x          BYTE    2
y          BYTE    3
z          BYTE    ?

.code
LOAD  A, [x]
LOAD  B, [y]
MOVE  C, A      ; z=x;
ADD   C, B      ; z+=y;
STORE [z], C
```

User Input from the Switches (arithmetic example)



C vs. Assembly

```
#include<stdio.h>

int x = 2;
int y;
int result;

// Addition with User input
// for the second variable (y)

void main()
{
    scanf("%d", &y);

    result = x + y;

    //printf("%d", result);
}
```

```
; Arithmetic with user input from switches
; Assembly version

.data
x            BYTE 2
y            BYTE ?
result       BYTE ?

.code

        LOAD    A, [x]            ; Reg. A <- [x]
        INPUTD [y]                ; [y] <- Input
        LOAD    B, [y]            ; Reg. B <- [y]
        ADD     A, B                ; A <- A + B
        STORE   [result], A        ; result <- Reg A
```

C vs. Assembly

```
#include<stdio.h>

int x = 2;
int y;
int result;

// Addition with User input
// for the second variable (y)

void main()
{
    scanf("%d", &y);

    result = x + y;

    //printf("%d", result);
}
```

```
; Arithmetic with user input from switches
; Assembly version

.data
x            BYTE 2
y            BYTE ?
result       BYTE ?

.code

LOAD    A, [x]            ; Reg. A <- [x]
INPUTD  [y]              ; [y] <- Input
LOAD    B, [y]            ; Reg. B <- [y]
ADD     A, B              ; A <- A + B
STORE   [result], A      ; result <- Reg A
```

Read the current value from switches SW07 to SW01 and store this value in the DMEM cell with label y.

Multiplication

C Version

```
// Multiplication  
//  
// C version
```

```
int main()  
{  
    int x=2;  
    int z;  
  
    z = x*5;  
  
    // printf("%d\n", z);  
}
```

Assembly Version

```
; Multiplication
```

```
.data
```

```
x          BYTE          3
z          BYTE          ?
```

```
.code
```

```
    LOAD    A, [x]
    MOVE    C, A          ; z=x;
    MOVE    B, A          ; B=x;
    SHIFTL  B            ; B=2x
    SHIFTL  B            ; B=4x
    ADD     C, B          ; C=4x+x
    STORE   [z], C       ; update the memory for z
```

```
; Register allocation
```

```
;
```

```
; A: x
```

```
; B: temporary results for 2x and 4x
```

```
; C: z
```

```
; D: <not used>
```

C vs. Assembly

```
// Multiplication
//
// C version

int main()
{
    int x=2;
    int z;

    z = x*5;

    // printf("%d\n", z);
}
```

```
; Multiplication

.data
x      BYTE    3
z      BYTE    ?

.code

    LOAD     A, [x]
    MOVE    C, A      ; z=x;
    MOVE    B, A      ; B=x;
    SHIFTL  B        ; B=2x
    SHIFTL  B        ; B=4x
    ADD     C, B      ; C=4x+x
    STORE   [z], C    ; update the
                    ; memory for z
```

Multiplication

(implemented with repeated addition)

C Version

```
// Multiplication  
//  
// C version
```

```
int main()  
{  
    int x=2;  
    int z;  
  
    z = x*5;  
  
    // printf("%d\n", z);  
}
```

C Version

```
// Multiplication  
//  
// C version
```

```
int main()  
{  
    int x=3;  
    int z=0;  
  
    for(int i=0; i<5; i++){  
        z+=x;  
    }  
    // printf("%d\n", z);  
}
```

Assembly Version

```
; Assembly version
```

```
;
```

```
; The CPU does not have OPCODEs for multiplication.
```

```
; Therefore, it is emulated with repeated addition.
```

```
.data
```

```
x          BYTE          3
```

```
z          BYTE          ?
```

```
.code
```

```
    LOAD  A, [x]
```

```
    LOADI C, 0          ; z=0
```

```
    LOADI B, 0          ; i=0
```

```
    LOADI D, 5          ; sentinel value
```

```
For:  CMP  B, D          ; i<5?
```

```
    BRGE  End           ; if(i>=5), exit for loop
```

```
    ADD   C, A          ; z+=x
```

```
    ADDI  B, 1          ; i++
```

```
    JUMP  For           ; jump to For loop
```

```
End:  STORE [z], C      ; update the z value in memory
```

Assembly Version

```
; Assembly version
;
; The CPU does not have OPCODEs for multiplication.
; Therefore, it is emulated with repeated addition.
```

```
.data
x      BYTE      3
z      BYTE      ?

.code

      LOAD  A, [x]
      LOADI C, 0      ; z=0
      LOADI B, 0      ; i=0
      LOADI D, 5      ; sentinel value
For:   CMP   B, D      ; i<5?
      BRGE  End       ; if(i>=5), exit for loop
      ADD  C, A       ; z+=x
      ADDI B, 1       ; i++
      JUMP For        ; jump to For loop
End:   STORE [z], C   ; update the z value in memory
```

```
; Register allocation
;
; A: x
; B: i (optimized to register)
; C: z
; D: 5
```

C vs. Assembly

```
// Multiplication
//
// C version

int main()
{
    int x=3;
    int z=0;

    for(int i=0; i<5; i++){
        z+=x;
    }
    // printf("%d\n", z);
}
```

```
; Assembly version

.data
x          BYTE    3
z          BYTE    ?

.code

        LOAD  A, [x]
        LOADI C, 0      ; z=0
        LOADI B, 0      ; i=0
        LOADI D, 5      ; sentinel value
For:    CMP    B, D      ; i<5?
        BRGE  End      ; if(i>=5), exit loop
        ADD  C, A      ; z+=x
        ADDI B, 1      ; i++
        JUMP For      ; jump to For loop
End:    STORE [z], C    ; update the z
                          ; value in memory
```

**If Statement:
Comparison of Signed Numbers**

The i281 Assembly Instructions

NOOP	NO Operation
INPUTC	INPUT into Code memory
INPUTCF	INPUT into Code memory with offset
INPUTD	INPUT into Data memory
INPUTDF	INPUT into Data memory with offset
MOVE	MOVE the contents of one register into another
LOADI	LOAD Immediate value
LOADP	LOAD Pointer address
ADD	ADD two registers
ADDI	ADD an Immediate value to a register
SUB	SUBtract two registers
SUBI	SUBtract an Immediate value from a register
LOAD	LOAD from a data memory address into a register
LOADF	LOAD with an offset specified by another register
STORE	STORE a register into a data memory address
STOREF	STORE with an offset specified by another register
SHIFTL	SHIFT Left all bits in a register
SHIFTR	SHIFT Right all bits in a register
CMP	COMPare the values in two registers
JUMP	JUMP unconditionally to a specified address
BRE	BRanch if Equal
BRZ	BRanch if Zero
BRNE	BRanch if Not Equal
BRNZ	BRanch if Not Zero
BRG	BRanch if Greater
BRGE	BRanch if Greater than or Equal

The i281 Assembly Instructions

NOOP	NO Operation	
INPUTC	INPUT into Code memory	
INPUTCF	INPUT into Code memory with offset	
INPUTD	INPUT into Data memory	
INPUTDF	INPUT into Data memory with offset	
MOVE	MOVE the contents of one register into another	
LOADI	LOAD Immediate value	
LOADP	LOAD Pointer address	
ADD	ADD two registers	
ADDI	ADD an Immediate value to a register	
SUB	SUBtract two registers	
SUBI	SUBtract an Immediate value from a register	
LOAD	LOAD from a data memory address into a register	
LOADF	LOAD with an offset specified by another register	
STORE	STORE a register into a data memory address	
STOREF	STORE with an offset specified by another register	
SHIFTL	SHIFT Left all bits in a register	
SHIFTR	SHIFT Right all bits in a register	
CMP	COMPare the values in two registers	
JUMP	JUMP unconditionally to a specified address	
BRE	BRanch if Equal	These 6 are used to implement comparisons
BRZ	BRanch if Zero	
BRNE	BRanch if Not Equal	
BRNZ	BRanch if Not Zero	
BRG	BRanch if Greater	
BRGE	BRanch if Greater than or Equal	

The i281 Assembly Instructions

NOOP	NO Operation
INPUTC	INPUT into Code memory
INPUTCF	INPUT into Code memory with offset
INPUTD	INPUT into Data memory
INPUTDF	INPUT into Data memory with offset
MOVE	MOVE the contents of one register into another
LOADI	LOAD Immediate value
LOADP	LOAD Pointer address
ADD	ADD two registers
ADDI	ADD an Immediate value to a register
SUB	SUBtract two registers
SUBI	SUBtract an Immediate value from a register
LOAD	LOAD from a data memory address into a register
LOADF	LOAD with an offset specified by another register
STORE	STORE a register into a data memory address
STOREF	STORE with an offset specified by another register
SHIFTL	SHIFT Left all bits in a register
SHIFTR	SHIFT Right all bits in a register
CMP	COMPare the values in two registers
JUMP	JUMP unconditionally to a specified address
BRE	BRanch if Equal
BRZ	BRanch if Zero
BRNE	BRanch if Not Equal
BRNZ	BRanch if Not Zero
BRG	BRanch if Greater
BRGE	BRanch if Greater than or Equal
BRL	BRanch if Less
BRLE	BRanch if Less than or Equal

These two instructions
are NOT supported
by this CPU

The i281 Assembly Instructions

NOOP	NO Operation
INPUTC	INPUT into Code memory
INPUTCF	INPUT into Code memory with offset
INPUTD	INPUT into Data memory
INPUTDF	INPUT into Data memory with offset
MOVE	MOVE the contents of one register into another
LOADI	LOAD Immediate value
LOADP	LOAD Pointer address
ADD	ADD two registers
ADDI	ADD an Immediate value to a register
SUB	SUBtract two registers
SUBI	SUBtract an Immediate value from a register
LOAD	LOAD from a data memory address into a register
LOADF	LOAD with an offset specified by another register
STORE	STORE a register into a data memory address
STOREF	STORE with an offset specified by another register
SHIFTL	SHIFT Left all bits in a register
SHIFTR	SHIFT Right all bits in a register
CMP	COMPare the values in two registers
JUMP	JUMP unconditionally to a specified address
BRE	BRanch if Equal
BRZ	BRanch if Zero
BRNE	BRanch if Not Equal
BRNZ	BRanch if Not Zero
BRG	BRanch if Greater
BRGE	BRanch if Greater than or Equal
BRL	BRanch if Less
BRLE	BRanch if Less than or Equal

comparison
of signed
numbers
only

Common Comparisons

`(x < y)`

`(x <= y)`

`(x == y)`

`(x != y)`

`(x >= y)`

`(x > y)`

Common Comparisons

`(x < y)`

`(x <= y)`

`(x == y)`

`(x != y)`

`(x >= y)`

`(x > y)`

`CMP x,y`

`BRGE End`

`CMP x,y`

`BRG End`

`CMP x,y`

`BRNE End`

`CMP x,y`

`BRE End`

`CMP x,y`

`BRL End`

`CMP x,y`

`BRLE End`

Where `x` and `y` are two registers.

Common Comparisons

$(x < y)$

$(x \leq y)$

$(x == y)$

$(x \neq y)$

$(x \geq y)$

$(x > y)$

CMP x,y
BRGE End

CMP x,y
BRG End

CMP x,y
BRNE End

CMP x,y
BRE End

CMP x,y
BRL End

CMP x,y
BRLE End

These two instructions
are NOT supported
by this CPU.

Common Comparisons

Therefore, invert the order.

$(x < y)$

$(x \leq y)$

$(x == y)$

$(x \neq y)$

$(y \leq x)$

$(y < x)$

~~$(x \geq y)$~~

~~$(x > y)$~~

CMP x,y
BRGE End

CMP x,y
BRG End

CMP x,y
BRNE End

CMP x,y
BRE End

CMP x,y
BRL End

CMP x,y
BRLE End

Common Comparisons

$(x < y)$

$(x \leq y)$

$(x == y)$

$(x \neq y)$

CMP x,y
BRGE End

CMP x,y
BRG End

CMP x,y
BRNE End

CMP x,y
BRE End

Therefore, invert the order.

~~$(y \leq x)$ $(y < x)$~~

~~$(x \geq y)$ $(x > y)$~~

~~**CMP x,y** **CMP x,y**
BRL End **BRLE End**~~

CMP y,x **CMP y,x**
BRG End **BRGE End**

If Less

C vs. Assembly

```
// If Less
//
// C version

int main()
{

    int x=3;
    int y=5;
    int z=0;

    if(x < y)
        z=x;
}
```

```
; If Less
;
; Assembly version

.data
x      BYTE    3
y      BYTE    5
z      BYTE    0

.code

    LOAD  A, [x]
    LOAD  B, [y]
    CMP   A, B
    BRGE  End
    STORE [z], A
End:    NOOP
```

C vs. Assembly

```
// If Less
//
// C version

int main()
{

    int x=3;
    int y=5;
    int z=0;

    if(x < y)
        z=x;

}
```

```
; If Less
;
; Assembly version

.data
x      BYTE    3
y      BYTE    5
z      BYTE    0

.code

    LOAD  A, [x]
    LOAD  B, [y]
    CMP   A, B
    BRGE  End
    STORE [z], A

End:   NOOP
```

instead of $x < y$,
check if $x \geq y$

If Less or Equal

C vs. Assembly

```
// If Less or Equal
//
// C version
```

```
int main()
{
    int x=3;
    int y=5;
    int z=0;

    if(x <= y)
        z=x;
}
```

```
; If Less or Equal
;
; Assembly version
```

```
.data
x      BYTE    3
y      BYTE    5
z      BYTE    0

.code

    LOAD  A, [x]
    LOAD  B, [y]
    CMP   A, B
    BRG   End
    STORE [z], A

End:   NOOP
```

C vs. Assembly

```
// If Less or Equal
//
// C version
```

```
int main()
{
    int x=3;
    int y=5;
    int z=0;

    if(x <= y)
        z=x;
}
```

```
; If Less or Equal
;
; Assembly version
```

.data

```
x      BYTE    3
y      BYTE    5
z      BYTE    0
```

.code

```
LOAD  A, [x]
LOAD  B, [y]
CMP  A, B
BRG  End
STORE [z], A
End:  NOOP
```

instead of $x \leq y$,
check if $x > y$

If Equal

C vs. Assembly

```
// If Equal
//
// C version
```

```
int main()
{
```

```
    int x=3;
    int y=5;
    int z=0;
```

```
    if(x==y)
        z=x;
```

```
}
```

```
; If Equal
;
; Assembly version
```

```
.data
```

```
x      BYTE  3
y      BYTE  5
z      BYTE  0
```

```
.code
```

```
LOAD  A, [x]
LOAD  B, [y]
CMP   A, B
BRNE  End
STORE [z], A
```

```
End:  NOOP
```

C vs. Assembly

```
// If Equal
//
// C version

int main()
{

    int x=3;
    int y=5;
    int z=0;

    if(x==y)
        z=x;

}
```

```
; If Equal
;
; Assembly version

.data
x      BYTE  3
y      BYTE  5
z      BYTE  0

.code

    LOAD  A, [x]
    LOAD  B, [y]
    CMP   A, B
    BRNE  End
    STORE [z], A

End:   NOOP
```

instead of `x == y`,
check if `x != y`

If Not Equal

C vs. Assembly

```
// If Not Equal
//
// C version
```

```
int main()
{
    int x=3;
    int y=5;
    int z=0;

    if(x != y)
        z=x;
}
```

```
; If Not Equal
;
; Assembly version
```

.data

```
x      BYTE    3
y      BYTE    5
z      BYTE    0
```

.code

```
LOAD  A, [x]
LOAD  B, [y]
CMP   A, B
BRE   End
STORE [z], A
End:  NOOP
```

C vs. Assembly

```
// If Not Equal
//
// C version
```

```
int main()
{
    int x=3;
    int y=5;
    int z=0;

    if(x != y)
        z=x;
}
```

```
; If Not Equal
;
; Assembly version
```

.data

```
x      BYTE  3
y      BYTE  5
z      BYTE  0
```

.code

```
LOAD  A, [x]
LOAD  B, [y]
CMP  A, B
BRE  End
STORE [z], A
End:  NOOP
```

instead of `x != y`,
check if `x == y`

If Greater or Equal

C vs. Assembly

```
// If Greater or Equal
//
// C version
```

```
int main()
{
    int x=3;
    int y=5;
    int z=0;

    if(x >= y)
        z=x;
}
```

```
; If Greater or Equal
;
; Assembly version
```

.data

```
x      BYTE    3
y      BYTE    5
z      BYTE    0
```

.code

```
LOAD   A, [x]
LOAD   B, [y]
CMP    B, A      ; these are swapped
BRG    End
STORE  [z], A
End:   NOOP
```

C vs. Assembly

```
// If Greater or Equal
//
// C version
```

```
int main()
{
    int x=3;
    int y=5;
    int z=0;

    if(y <= x)
        z=x;
}
```

```
; If Greater or Equal
;
; Assembly version
```

.data

```
x      BYTE    3
y      BYTE    5
z      BYTE    0
```

.code

```
LOAD  A, [x]
LOAD  B, [y]
CMP   B, A
BRG   End
STORE [z], A
End:  NOOP
```

; these are swapped

C vs. Assembly

```
// If Greater or Equal
//
// C version
```

```
int main()
{
    int x=3;
    int y=5;
    int z=0;

    if(y <= x)
        z=x;
}
```

```
; If Greater or Equal
;
; Assembly version
```

.data

```
x      BYTE    3
y      BYTE    5
z      BYTE    0
```

.code

```
LOAD  A, [x]
LOAD  B, [y]
CMP  B, A
BRG  End
STORE [z], A
End:  NOOP
```

; these are swapped

instead of $y \leq x$,
check if $y > x$

If Greater

C vs. Assembly

```
// If Greater
//
// C version
```

```
int main()
{
    int x=3;
    int y=5;
    int z=0;

    if(x > y)
        z=x;
}
```

```
; If Greater
;
; Assembly version
```

.data

```
x      BYTE    3
y      BYTE    5
z      BYTE    0
```

.code

```
LOAD   A, [x]
LOAD   B, [y]
CMP    B, A      ; these are swapped
BRGE   End
STORE  [z], A
End:   NOOP
```

C vs. Assembly

```
// If Greater
//
// C version
```

```
int main()
{
    int x=3;
    int y=5;
    int z=0;

    if(y < x)
        z=x;
}
```

```
; If Greater
;
; Assembly version
```

.data

```
x      BYTE    3
y      BYTE    5
z      BYTE    0
```

.code

```
LOAD   A, [x]
LOAD   B, [y]
CMP    B, A
BRGE   End
STORE  [z], A
End:   NOOP
```

; these are swapped

C vs. Assembly

```
// If Greater
//
// C version
```

```
int main()
{
    int x=3;
    int y=5;
    int z=0;

    if(y < x)
        z=x;
}
```

```
; If Greater
;
; Assembly version
```

.data

```
x      BYTE    3
y      BYTE    5
z      BYTE    0
```

.code

```
LOAD  A, [x]
LOAD  B, [y]
CMP   B, A
BRGE End
STORE [z], A
End:  NOOP
```

; these are swapped

instead of $y < x$,
check if $y \geq x$

Common Comparisons

$(x < y)$

$(x \leq y)$

$(x == y)$

$(x \neq y)$

CMP x,y
BRGE End

CMP x,y
BRG End

CMP x,y
BRNE End

CMP x,y
BRE End

~~$(y \leq x)$ $(y < x)$~~

~~$(x \geq y)$ $(x > y)$~~

~~**CMP x,y** **CMP x,y**
BRL End **BRLE End**~~

CMP y,x **CMP y,x**
BRG End **BRGE End**

**If Statement:
Comparison of **Unsigned** Numbers**

(not supported by this CPU)

Comparison of Signed Numbers

- Equal ZF
- Not equal \overline{ZF}
- Greater $\overline{ZF} \cdot XNOR(NF, OF)$
- Greater or Equal $XNOR(NF, OF)$
- Less $XOR(NF, OF)$
- Less or Equal $ZF + XOR(NF, OF)$

Comparison of **Unsigned** Numbers

- Equal ZF
- Not equal \overline{ZF}
- Above $\overline{ZF} \cdot CF$
- Above or Equal CF
- Below \overline{CF}
- Below or Equal $ZF + \overline{CF}$

Signed v.s. Unsigned

- Equal ZF
- Not equal \overline{ZF}
- Greater $\overline{ZF} \cdot XNOR(NF, OF)$
- Greater or Equal $XNOR(NF, OF)$
- Less $XOR(NF, OF)$
- Less or Equal $ZF + XOR(NF, OF)$

- Equal ZF
- Not equal \overline{ZF}
- Above $\overline{ZF} \cdot CF$
- Above or Equal CF
- Below \overline{CF}
- Below or Equal $ZF + \overline{CF}$

Signed v.s. Unsigned

they overlap only here

• Equal

ZF

• Not equal

$\overline{\text{ZF}}$

• Greater

$\overline{\text{ZF}} \cdot \text{XNOR}(\text{NF}, \text{OF})$

• Greater or Equal

$\text{XNOR}(\text{NF}, \text{OF})$

• Less

$\text{XOR}(\text{NF}, \text{OF})$

• Less or Equal

$\text{ZF} + \text{XOR}(\text{NF}, \text{OF})$

• Equal

ZF

• Not equal

$\overline{\text{ZF}}$

• Above

$\overline{\text{ZF}} \cdot \text{CF}$

• Above or Equal

CF

• Below

$\overline{\text{CF}}$

• Below or Equal

$\text{ZF} + \overline{\text{CF}}$

Signed v.s. Unsigned

BRE	BRanch if Equal
BRZ	BRanch if Zero
BRNE	BRanch if Not Equal
BRNZ	BRanch if Not Zero
BRG	BRanch if Greater
BRGE	BRanch if Greater than or Equal
BRL	BRanch if Less
BRLE	BRanch if Less than or Equal

BRE	Ranch if Equal
BRZ	BRanch if Zero
BRNE	BRanch if Not Equal
BRNZ	BRanch if Not Zero
BRA	BRanch if Above
BRAE	BRanch if Above or Equal
BRB	BRanch if Below
BRBE	BRanch if Below or Equal

Signed v.s. Unsigned

BRE BRanch if Equal
BRZ BRanch if Zero

BRNE BRanch if Not Equal
BRNZ BRanch if Not Zero

BRG BRanch if Greater

BRGE BRanch if Greater than or Equal

BRL BRanch if Less

BRLE BRanch if Less than or Equal

BRE Ranch if Equal
BRZ BRanch if Zero

BRNE BRanch if Not Equal
BRNZ BRanch if Not Zero

BRA BRanch if Above

BRAE BRanch if Above or Equal

BRB BRanch if Below

BRBE BRanch if Below or Equal

The OPCODEs in blue
are NOT implemented
on this CPU

Thus, comparing unsigned numbers requires adding **new** OPCODEs

BRE	Branch if Equal
BRZ	Branch if Zero
BRNE	Branch if Not Equal
BRNZ	Branch if Not Zero
BRA	Branch if Above
BRAE	Branch if Above or Equal
BRB	Branch if Below
BRBE	Branch if Below or Equal

Thus, comparing unsigned numbers requires adding **new** OPCODEs

BRE	Branch if Equal
BRZ	Branch if Zero
BRNE	Branch if Not Equal
BRNZ	Branch if Not Zero
BRA	Branch if Above
BRAE	Branch if Above or Equal
BRB	Branch if Below
BRBE	Branch if Below or Equal

at a minimum
you need only
these two



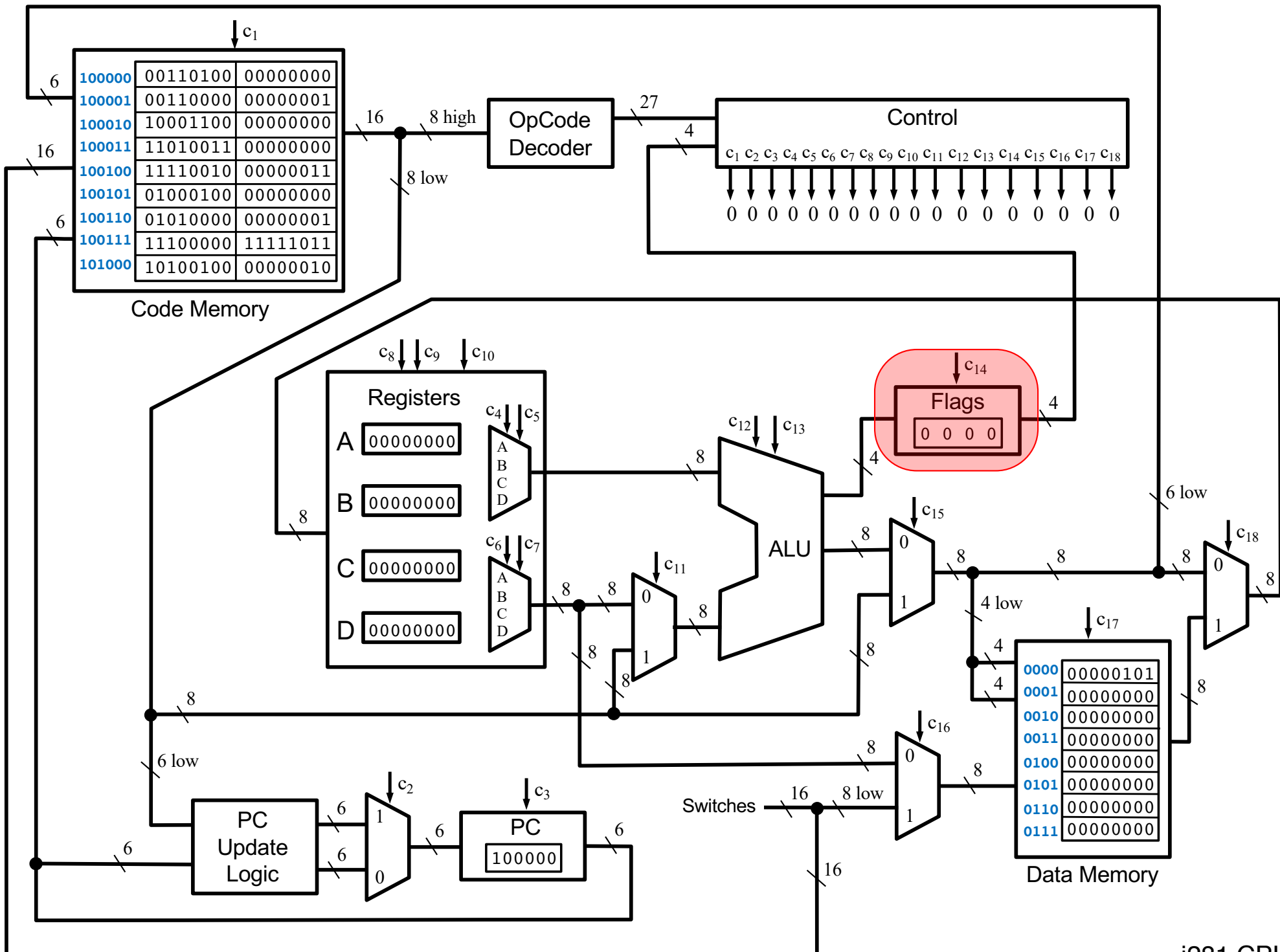
Thus, comparing unsigned numbers requires adding **new** OPCODEs

BRE	Branch if Equal
BRZ	Branch if Zero
BRNE	Branch if Not Equal
BRNZ	Branch if Not Zero
BRA	Branch if Above
BRAE	Branch if Above or Equal
BRB	Branch if Below
BRBE	Branch if Below or Equal

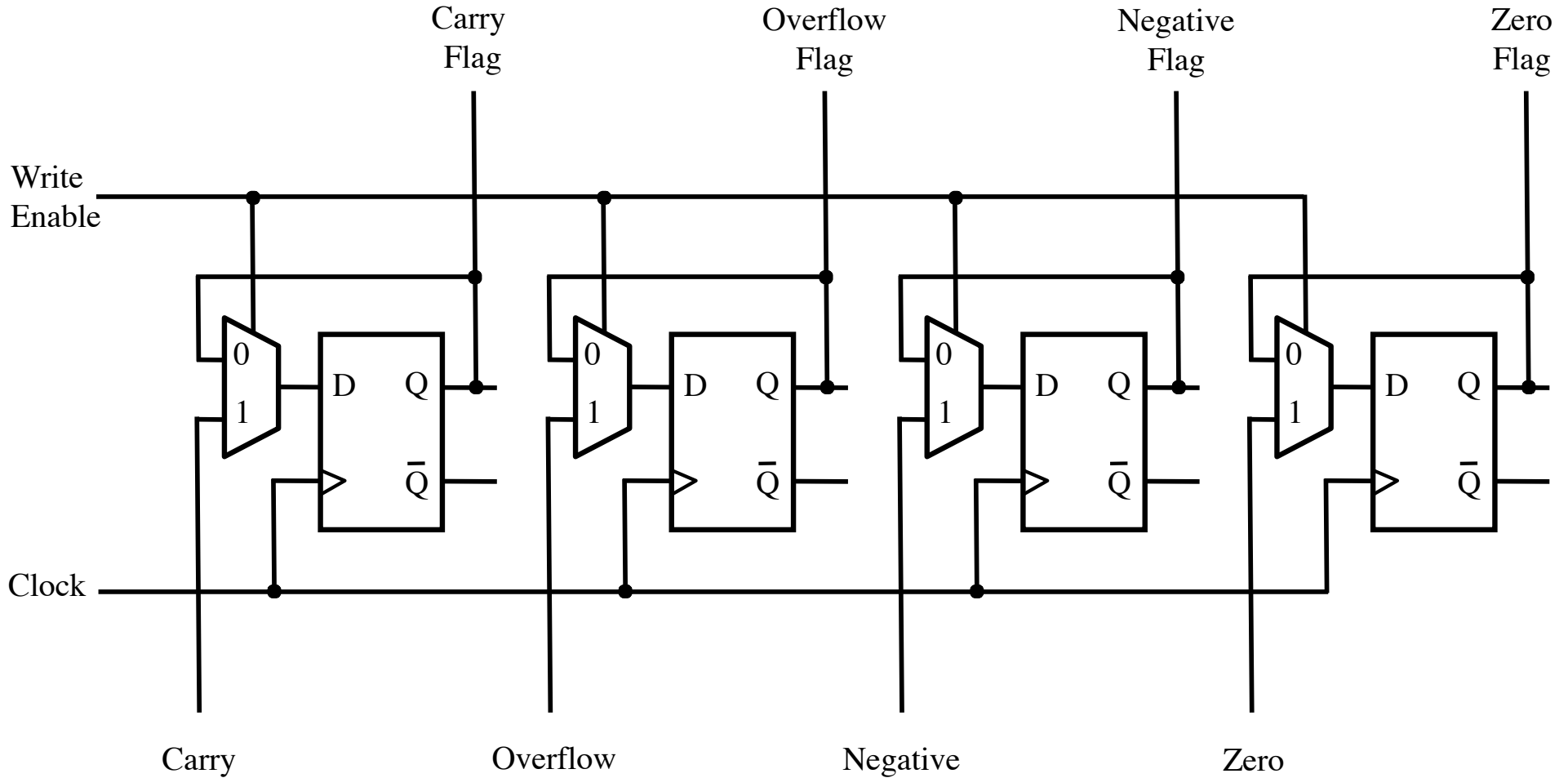
or these two



OPCODEs that check the Flags



The Flags Register



You could also add OPCODEs that just check a specific flag

BRC	BRanch if Carry
BRNC	BRanch if No Carry
BRO	BRanch if Overflow
BRNO	BRanch if No Overflow
BRN	BRanch if Negative
BRNN	BRanch if Not Negative

BRZ	BRanch if Zero
BRNZ	BRanch if Not Zero

} only these two
are currently
supported

You could also add OPCODEs that just check a specific flag

BRC BRanch if Carry

BRNC BRanch if No Carry

BRO BRanch if Overflow

BRNO BRanch if No Overflow

BRN BRanch if Negative

BRNN BRanch if Not Negative

BRZ BRanch if Zero

BRNZ BRanch if Not Zero

} BRE

BRNE

equivalent to

You could also add OPCODEs that just check a specific flag

		equivalent to
BRC	BRanch if Carry	} BRAE BRB
BRNC	BRanch if No Carry	
BRO	BRanch if Overflow	
BRNO	BRanch if No Overflow	
BRN	BRanch if Negative	
BRNN	BRanch if Not Negative	
BRZ	BRanch if Zero	
BRNZ	BRanch if Not Zero	

Signed v.s. Unsigned

- Equal ZF
- Not equal $\overline{\text{ZF}}$

- Greater $\overline{\text{ZF}} \cdot \text{XNOR}(\text{NF}, \text{OF})$
- Greater or Equal $\text{XNOR}(\text{NF}, \text{OF})$
- Less $\text{XOR}(\text{NF}, \text{OF})$
- Less or Equal $\text{ZF} + \text{XOR}(\text{NF}, \text{OF})$

- Equal ZF
- Not equal $\overline{\text{ZF}}$
- Above $\overline{\text{ZF}} \cdot \text{CF}$
- Above or Equal CF
- Below $\overline{\text{CF}}$
- Below or Equal $\text{ZF} + \overline{\text{CF}}$

How to add more BRanch OpCODEs

BRZ/BRN

1	1	1	1	d	d	0	0	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

BRNE/BRNZ

1	1	1	1	d	d	0	1	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

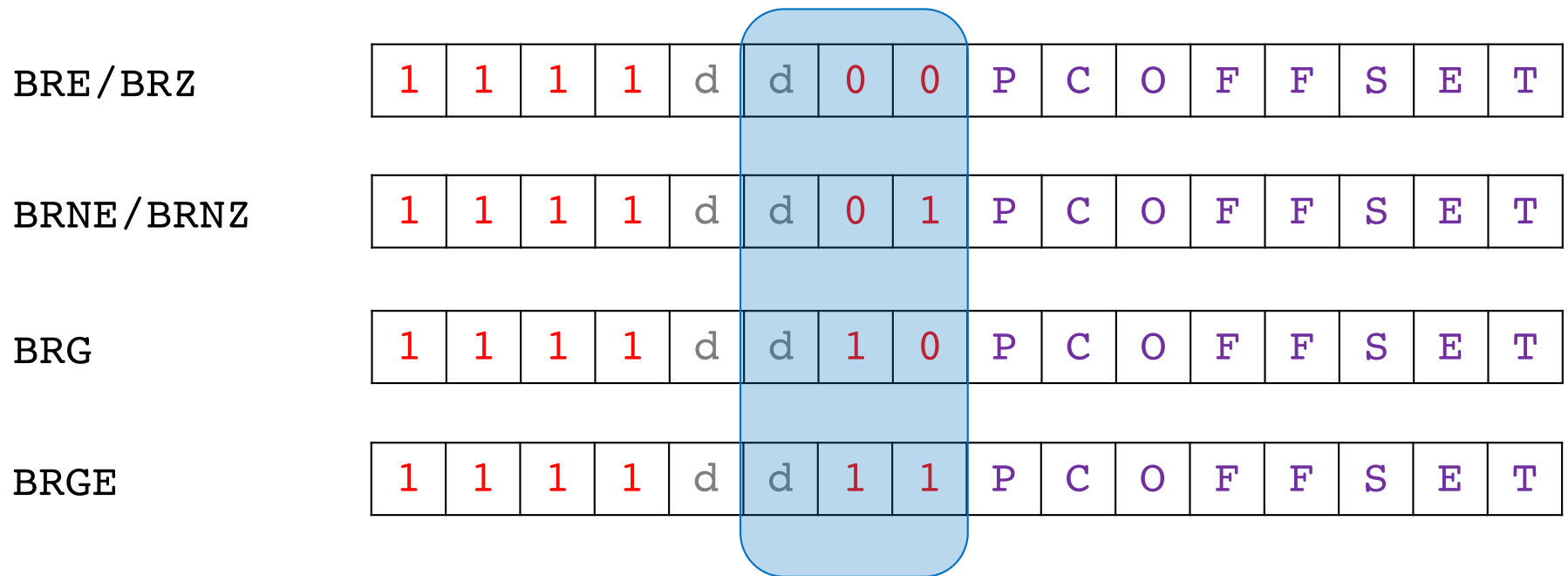
BRG

1	1	1	1	d	d	1	0	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

BRGE

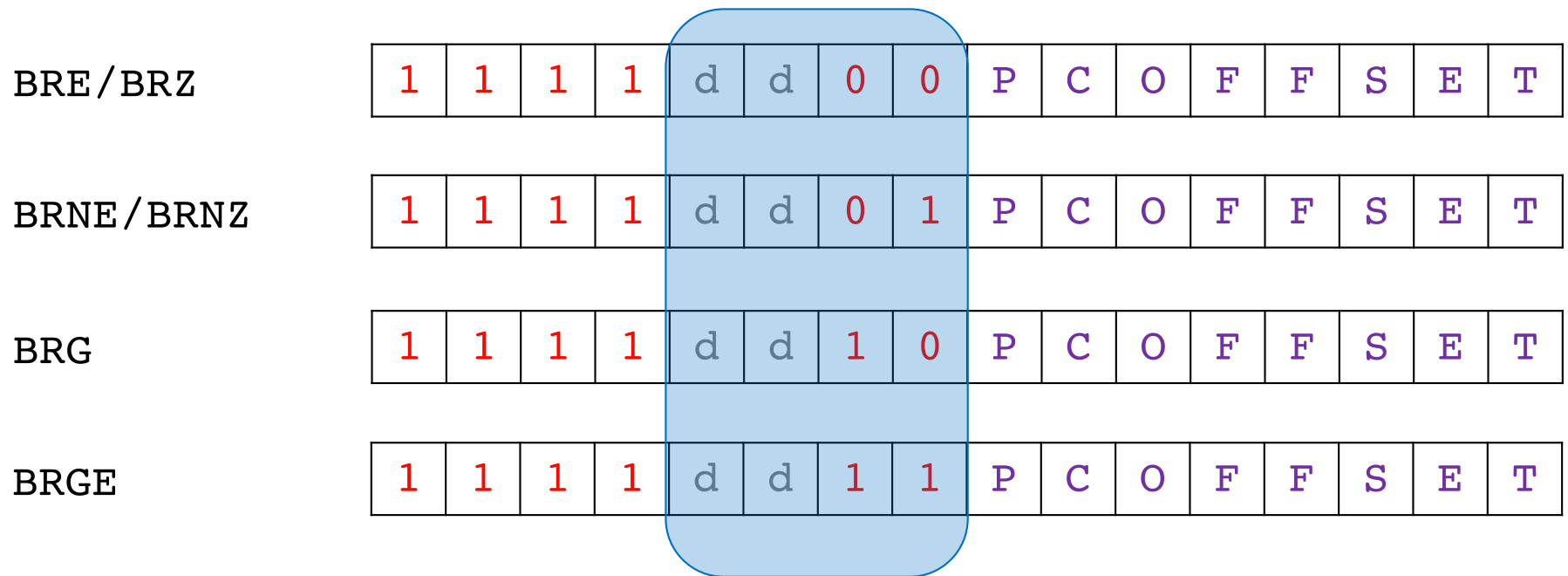
1	1	1	1	d	d	1	1	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

How to add more BRanch OpCODES

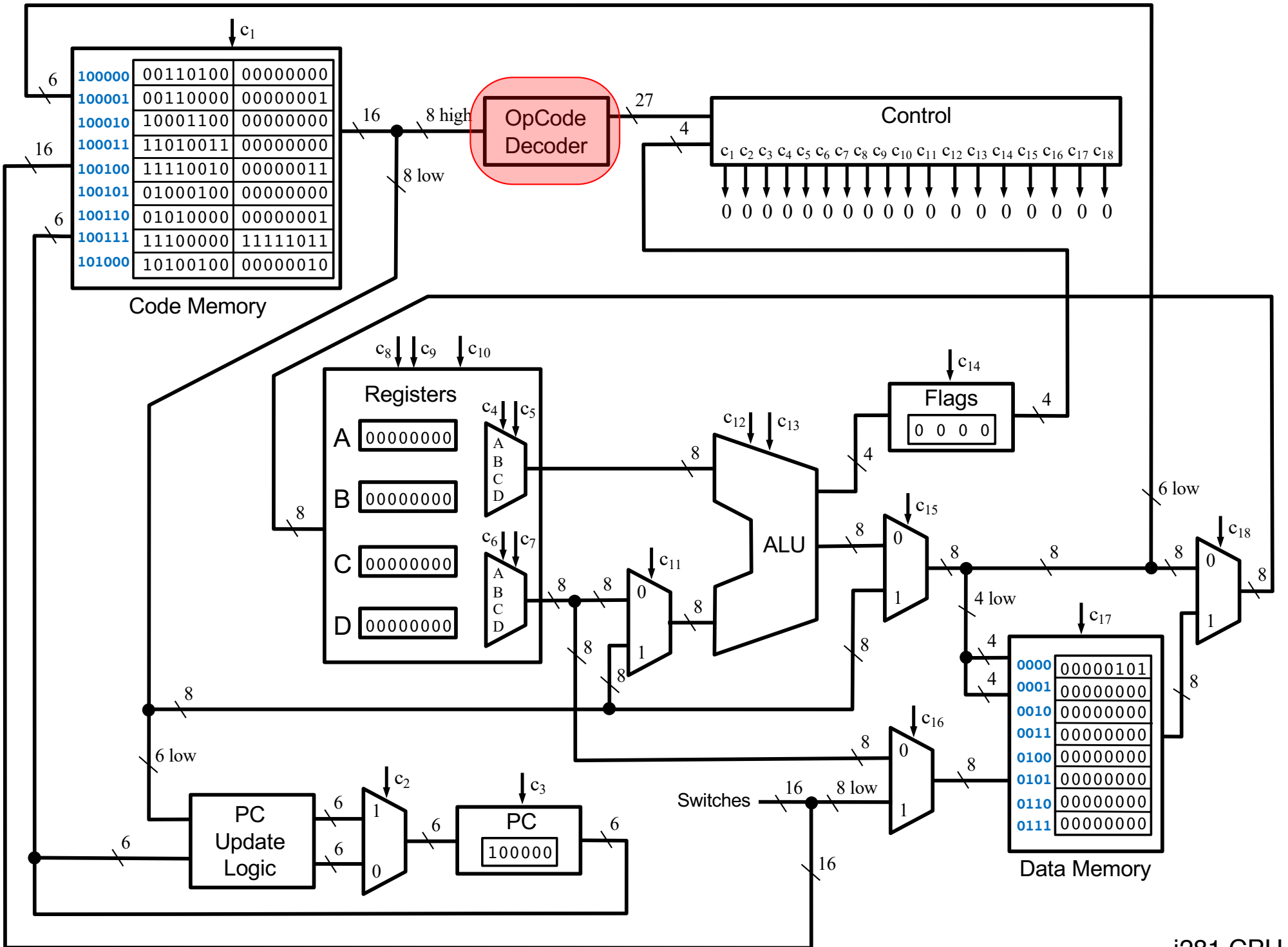


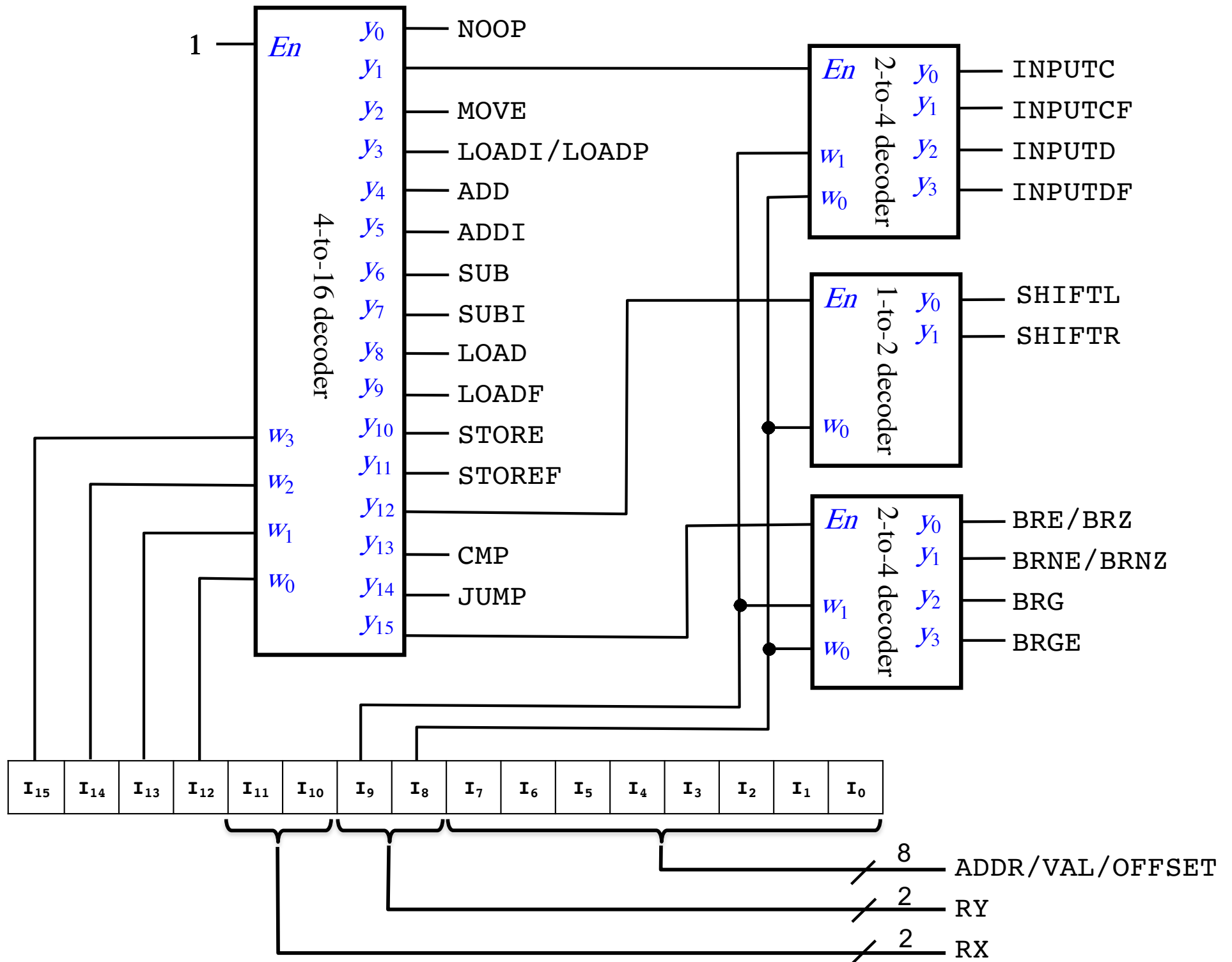
could use this don't care bit and
update the OpCODE decoder

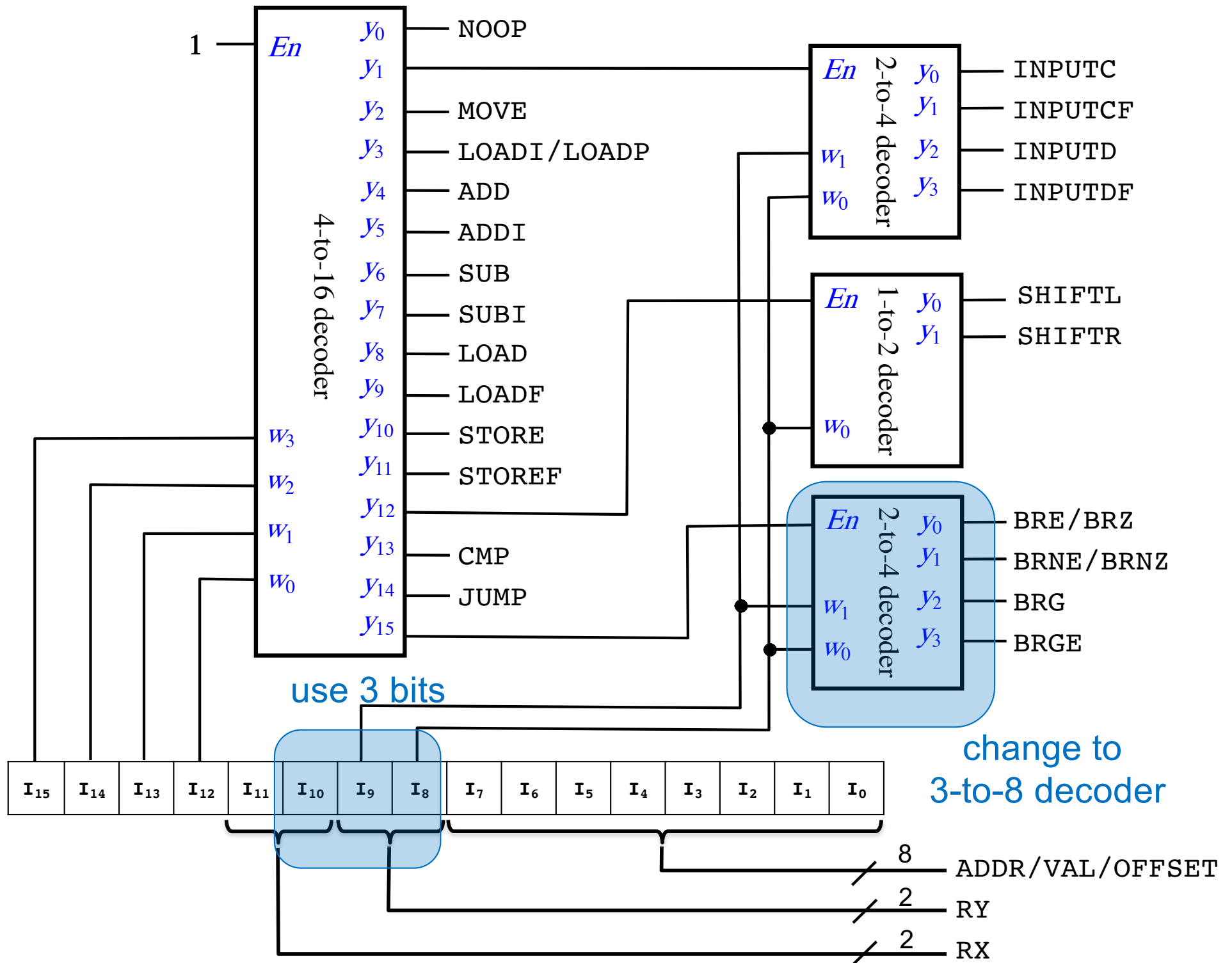
How to add more BRanch OpCODES

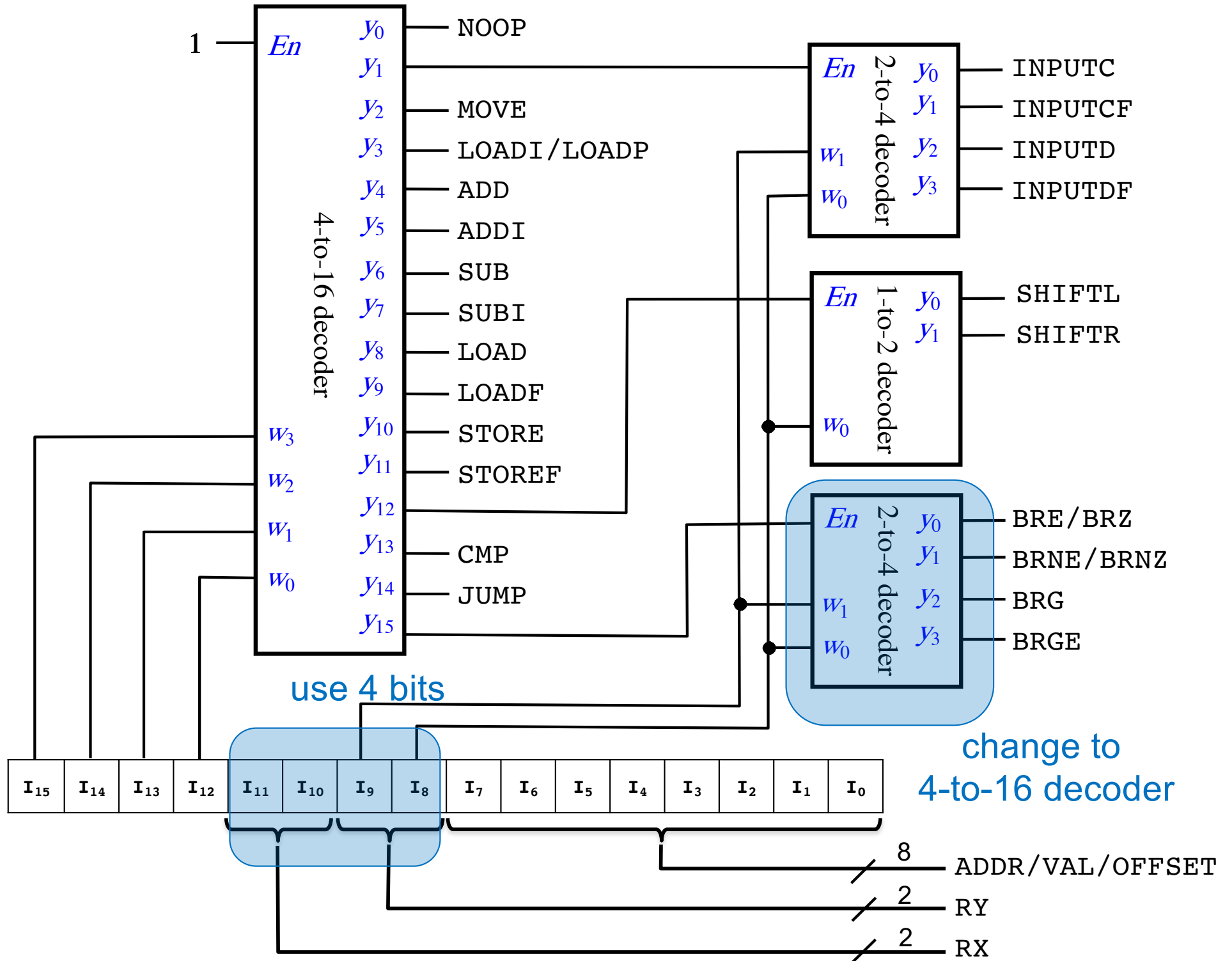


could use these two don't care bits and
update the OpCODE decoder









If-Else Statement

C vs. Assembly

```
// If-Else Statement
//
// C version
```

```
int main()
{
    int x=3;
    int y=5;
    int z;

    if (x<y)
        z=x;
    else
        z=y;
}
```

```
; If If-Else Statement
;
; Assembly version
```

```
.data
x      BYTE    3
y      BYTE    5
z      BYTE    ?

.code

        LOAD   A, [x]
        LOAD   B, [y]
If:     CMP    A, B
        BRGE   Else
        STORE  [z], A
        JUMP   End
Else:   STORE  [z], B
End:    NOOP
```

If with two conditions (OR)

C vs. Assembly

```
// If with 2 conditions (OR)
//
// C version

int main()
{

    int x=9;
    int min=1;
    int max=8;
    int inRange=1;

    if ( (x<min) || (x>max) )
        inRange=0;
}
```

```
; Assembly version
```

```
.data
```

```
x            BYTE    9
min          BYTE    1
max          BYTE    8
inRange     BYTE    1
```

```
.code
```

```
                LOAD   A, [x]
                LOAD   B, [min]
                LOAD   C, [max]
First:          CMP    A, B                ; x < min ?
                BRGE   Second
Set:            LOADI  D, 0
                STORE  [inRange], D
                JUMP   End
Second:        CMP    C, A                ; max < x ?
                BRGE   End
                JUMP   Set
End:           NOOP
```


C vs. Assembly

```
// If with 2 conditions (OR)
// this is closer to assembly
int main()
{
    int x=9;
    int min=1;
    int max=8;
    int inRange=1;

    if(x < min)    // First
    {
        Set:
            inRange=0;
            goto End;
    }
    else if(max < x) // Second
    {
        // swapped condition
        goto Set;
    }
}
```

```
; Assembly version

.data
x            BYTE    9
min         BYTE    1
max         BYTE    8
inRange     BYTE    1

.code

                LOAD  A, [x]
                LOAD  B, [min]
                LOAD  C, [max]
First:         CMP   A, B                ; x < min ?
                BRGE  Second
Set:          LOADI D, 0
                STORE [inRange], D
                JUMP  End
Second:      CMP   C, A                ; max < x ?
                BRGE  End
                JUMP  Set
End:         NOOP
```

C vs. Assembly

```
// If with 2 conditions (OR)
// this is closer to assembly
int main()
{
    int x=9;
    int min=1;
    int max=8;
    int inRange=1;

    if(x < min) // First
    {
        Set:
            inRange=0;
            goto End;
    }
    else if(max < x) // Second
    {
        goto Set;
    }
}
```

; Assembly version

.data

```
x          BYTE    9
min        BYTE    1
max        BYTE    8
inRange    BYTE    1
```

.code

```
                LOAD  A, [x]
                LOAD  B, [min]
                LOAD  C, [max]
First:          CMP   A, B
                BRGE  Second
Set:           LOADI D, 0
                STORE [inRange], D
                JUMP  End
Second:        CMP   C, A
                BRGE  End
                JUMP  Set
End:           NOOP
```

instead of $x < \text{min}$,
check if $x \geq \text{min}$

; $\text{max} < x$?

C vs. Assembly

```
// If with 2 conditions (OR)
// this is closer to assembly
int main()
{
    int x=9;
    int min=1;
    int max=8;
    int inRange=1;

    if(x < min) // First
    {
        Set:
            inRange=0;
            goto End;
    }
    else if(max < x) // Second
    {
        goto Set;
    }
}
```

; Assembly version

```
.data
x            BYTE    9
min          BYTE    1
max          BYTE    8
inRange      BYTE    1

.code

            LOAD    A, [x]
            LOAD    B, [min]
            LOAD    C, [max]
First:     CMP     A, B
            BRGE   Second

Set:       LOADI   D, 0
            STORE  [inRange], D
            JUMP   End

Second:    CMP     C, A
            BRGE   End

            JUMP   Set

End:       NOOP
```

instead of $\text{max} < \text{x}$,
check if $\text{max} \geq \text{x}$

If with two conditions (AND)

C vs. Assembly

```
// If with 2 conditions (AND)
```

```
//
```

```
// C version
```

```
int main()
```

```
{
```

```
    int x=5;
```

```
    int min=1;
```

```
    int max=8;
```

```
    int inRange=0;
```

```
    if ( (x>=min) && (x<=max) )
```

```
        inRange=1;
```

```
}
```

```
; Assembly version
```

```
.data
```

```
x          BYTE    5
```

```
min        BYTE    1
```

```
max        BYTE    8
```

```
inRange    BYTE    0
```

```
.code
```

```
LOAD A, [x]
```

```
LOAD B, [min]
```

```
LOAD C, [max]
```

```
CMP B, A ; min <= x ?
```

```
BRG End
```

```
CMP A, C ; x <= max ?
```

```
BRG End
```

```
LOADI D, 1
```

```
STORE [inRange], D
```

```
End: NOOP
```

C vs. Assembly

```
// If with 2 conditions (AND)
//
// C version

int main()
{
    int x=5;
    int min=1;
    int max=8;
    int inRange=0;

    if(min<=x) // swapped dir.
    {
        if(x<=max)
        {
            inRange=1;
        }
    }
}
```

```
; Assembly version

.data
x            BYTE    5
min          BYTE    1
max          BYTE    8
inRange     BYTE    0

.code
            LOAD    A, [x]
            LOAD    B, [min]
            LOAD    C, [max]
            CMP     B, A                ; min <= x ?
            BRG     End
            CMP     A, C                ; x <= max ?
            BRG     End
            LOADI   D, 1
            STORE   [inRange], D
End:        NOOP
```

C vs. Assembly

```
// If with 2 conditions (AND)
//
// C version

int main()
{
    int x=5;
    int min=1;
    int max=8;
    int inRange=0;

    if(min<=x) // swapped dir.
    {
        if(x<=max)
        {
            inRange=1;
        }
    }
}
```

; Assembly version

.data

```
x          BYTE    5
min        BYTE    1
max        BYTE    8
inRange    BYTE    0
```

.code

```
LOAD  A, [x]
LOAD  B, [min]
LOAD  C, [max]
CMP  B, A
BRG  End
CMP   A, C           ; x <= max ?
BRG   End
LOADI D, 1
STORE [inRange], D
End:  NOOP
```

instead of min <= x,
check if min > x

C vs. Assembly

```
// If with 2 conditions (AND)
//
// C version

int main()
{
    int x=5;
    int min=1;
    int max=8;
    int inRange=0;

    if(min<=x) // swapped dir.
    {
        if(x<=max)
        {
            inRange=1;
        }
    }
}
```

```
; Assembly version

.data
x            BYTE    5
min         BYTE    1
max         BYTE    8
inRange     BYTE    0

.code
            LOAD    A, [x]
            LOAD    B, [min]
            LOAD    C, [max]
            CMP     B, A
            BRG     End
            CMP     A, C
            BRG     End
            LOADI   D, 1
            STORE  [inRange], D

End:       NOOP
```

instead of $x \leq \max$,
check if $x > \max$

Switch Statement

C vs. Assembly

```
// Switch Statement
//
// C version
```

```
int main()
{
    int x=6;
    int y;

    switch(x)
    {
        case 0:    y=2*x;
                  break;

        case 1:    y=x+3;
                  break;

        case 2:    y=x-1;
                  break;

        default:   y=x/2;
                  break;
    }
}
```

```
; Assembly version
```

```
.data
x          BYTE    6
y          BYTE    ?

.code

Zero:      LOAD    A, [x]
           LOADI   C, 0
           CMP     A, C
           BRNE   One
           MOVE   B, A
           SHIFTL B
           JUMP   End
One:       LOADI   C, 1
           CMP     A, C
           BRNE   Two
           MOVE   B, A
           ADDI   B, 3
           JUMP   End
Two:       LOADI   C, 2
           CMP     A, C
           BRNE   Default
           MOVE   B, A
           SUBI   B, 1
           JUMP   End
Default:   MOVE   B, A
           SHIFTR B
End:       STORE   [y], B
           NOOP
```

C vs. Assembly

```
// Switch Statement
//
// C version

int main()
{
    int x=6;
    int y;

    switch(x)
    {
        case 0:      y=2*x;
                    break;

        case 1:      y=x+3;
                    break;

        case 2:      y=x-1;
                    break;

        default:     y=x/2;
                    break;
    }
}
```

```
; Assembly version
.data
x          BYTE    6
y          BYTE    ?

.code
          LOAD    A, [x]
Zero:    LOADI   C, 0
          CMP     A, C
          BRNE   One
          MOVE   B, A
          SHIFTL B
          JUMP   End
One:     LOADI   C, 1
          CMP     A, C
          BRNE   Two
          MOVE   B, A
          ADDI   B, 3
          JUMP   End
Two:    LOADI   C, 2
          CMP     A, C
          BRNE   Default
          MOVE   B, A
          SUBI   B, 1
          JUMP   End
Default: MOVE   B, A
          SHIFTR B
End:     STORE   [y], B
          NOOP
```

C vs. Assembly

```
// Switch Statement
//
// C version

int main()
{
    int x=6;
    int y;

    switch(x)
    {
        case 0:      y=2*x;
                    break;

        case 1:      y=x+3;
                    break;

        case 2:      y=x-1;
                    break;

        default:     y=x/2;
                    break;
    }
}
```

```
; Assembly version
.data
x          BYTE        6
y          BYTE        ?

.code
          LOAD        A, [x]
Zero:     LOADI       C, 0
          CMP         A, C
          BRNE       One
          MOVE       B, A
          SHIFTL    B
          JUMP      End
One:      LOADI       C, 1
          CMP         A, C
          BRNE       Two
          MOVE       B, A
          ADDI      B, 3
          JUMP      End
Two:      LOADI       C, 2
          CMP         A, C
          BRNE       Default
          MOVE       B, A
          SUBI      B, 1
          JUMP      End
Default:  MOVE       B, A
          SHIFTR    B
End:      STORE     [y], B
          NOOP
```

C vs. Assembly

```
// Switch Statement
//
// C version
```

```
int main()
{
    int x=6;
    int y;

    switch(x)
    {
        case 0:    y=2*x;
                  break;

        case 1:    y=x+3;
                  break;

        case 2:    y=x-1;
                  break;

        default:   y=x/2;
                  break;
    }
}
```

```
; Assembly version
```

```
.data
x          BYTE    6
y          BYTE    ?

.code

Zero:      LOAD    A, [x]
           LOADI   C, 0
           CMP     A, C
           BRNE   One
           MOVE   B, A
           SHIFTL B
           JUMP   End
One:       LOADI   C, 1
           CMP     A, C
           BRNE   Two
           MOVE   B, A
           ADDI   B, 3
           JUMP   End
Two:       LOADI   C, 2
           CMP     A, C
           BRNE   Default
           MOVE   B, A
           SUBI   B, 1
           JUMP   End
Default:   MOVE   B, A
           SHIFTR B
End:       STORE   [Y], B
           NOOP
```

C vs. Assembly

```
// Switch Statement
//
// C version
```

```
int main()
{
```

```
    int x=6;
    int y;
```

```
    switch(x)
    {
```

```
        case 0:      y=2*x;
                    break;
```

```
        case 1:      y=x+3;
                    break;
```

```
        case 2:      y=x-1;
                    break;
```

```
        default:    y=x/2;
                    break;
```

```
    }
```

```
}
```

```
; Assembly version
```

```
.data
```

```
x          BYTE    6
```

```
y          BYTE    ?
```

```
.code
```

```
LOAD      A, [x]
```

```
Zero:     LOADI   C, 0
```

```
CMP       A, C
```

```
BRNE     One
```

```
MOVE     B, A
```

```
SHIFTL   B
```

```
JUMP     End
```

```
One:      LOADI   C, 1
```

```
CMP       A, C
```

```
BRNE     Two
```

```
MOVE     B, A
```

```
ADDI     B, 3
```

```
JUMP     End
```

```
Two:      LOADI   C, 2
```

```
CMP       A, C
```

```
BRNE     Default
```

```
MOVE     B, A
```

```
SUBI     B, 1
```

```
JUMP     End
```

```
Default: MOVE     B, A
```

```
SHIFTR   B
```

```
End:     STORE   [y], B
```

```
NOOP
```



C vs. Assembly

```
// Switch Statement
//
// C version
```

```
int main()
{
    int x=6;
    int y;

    switch(x)
    {
        case 0:    y=2*x;
                  break;

        case 1:    y=x+3;
                  break;

        case 2:    y=x-1;
                  break;

        default:   y=x/2;
                  break;
    }
}
```

```
; Assembly version
```

```
.data
x          BYTE    6
y          BYTE    ?

.code
Zero:      LOAD    A, [x]
           LOADI   C, 0
           CMP     A, C
           BRNE   One
           MOVE    B, A
           SHIFTL  B
           JUMP   End
One:       LOADI   C, 1
           CMP     A, C
           BRNE   Two
           MOVE    B, A
           ADDI    B, 3
           JUMP   End
Two:      LOADI   C, 2
           CMP     A, C
           BRNE   Default
           MOVE    B, A
           SUBI    B, 1
           JUMP   End
Default:  MOVE    B, A
           SHIFTR  B
End:      STORE   [y], B
           NOOP
```

C vs. Assembly

```
// Switch Statement
//
// C version
```

```
int main()
{
    int x=6;
    int y;

    switch(x)
    {
        case 0:    y=2*x;
                  break;

        case 1:    y=x+3;
                  break;

        case 2:    y=x-1;
                  break;

        default:   y=x/2;
                  break;
    }
}
```

```
; Assembly version
```

```
.data
x          BYTE    6
y          BYTE    ?

.code

Zero:      LOAD    A, [x]
           LOADI   C, 0
           CMP     A, C
           BRNE   One
           MOVE   B, A
           SHIFTL B
           JUMP   End

One:       LOADI   C, 1
           CMP     A, C
           BRNE   Two
           MOVE   B, A
           ADDI   B, 3
           JUMP   End

Two:       LOADI   C, 2
           CMP     A, C
           BRNE   Default
           MOVE   B, A
           SUBI   B, 1
           JUMP   End

Default:   MOVE   B, A
           SHIFTR B

End:       STORE   [y], B
           NOOP
```


C vs. Assembly

```
// Switch Statement
//
// C version
```

```
int main()
{
    int x=6;
    int y;

    switch(x)
    {
        case 0:    y=2*x;
                  break;

        case 1:    y=x+3;
                  break;

        case 2:    y=x-1;
                  break;

        default:   y=x/2;
                  break;
    }
}
```

```
; Assembly version
```

```
.data
x          BYTE    6
y          BYTE    ?

.code

Zero:      LOAD    A, [x]
           LOADI   C, 0
           CMP     A, C
           BRNE   One
           MOVE   B, A
           SHIFTL B
           JUMP   End

One:       LOADI   C, 1
           CMP     A, C
           BRNE   Two
           MOVE   B, A
           ADDI   B, 3
           JUMP   End

Two:       LOADI   C, 2
           CMP     A, C
           BRNE   Default
           MOVE   B, A
           SUBI   B, 1
           JUMP   End

Default:   MOVE   B, A
           SHIFTR B
           STORE  [Y], B
           NOOP

End:
```

← not mapped to instruction;

just goes to next line

Working with Arrays

C vs. Assembly

```
// Array
//
// C version

int array[]={1, 2, 3, 4};

int main()
{
    array[0] = 0;

    array[1] += 5;

    array[2] -= 1;

    array[3] += array[2];
}
```

```
; Array
;
; Assembly version

.data
array    BYTE    1, 2, 3, 4

.code

    LOADI    A, 0
    STORE    [array + 0], A
    LOAD     B, [array + 1]
    ADDI     B, 5
    STORE    [array + 1], B
    LOAD     C, [array + 2]
    SUBI     C, 1
    STORE    [array + 2], C
    LOAD     D, [array + 3]
    ADD      D, C
    STORE    [array + 3], D
```

C vs. Assembly

```
// Array
//
// C version
```

```
int array[]={1, 2, 3, 4};
```

```
int main()
```

```
{
```

```
    array[0] = 0;
```

```
    array[1] += 5;
```

```
    array[2] -= 1;
```

```
    array[3] += array[2];
```

```
}
```

```
; Array
;
; Assembly version
```

```
.data
```

```
array    BYTE    1, 2, 3, 4
```

```
.code
```

```
LOADI    A, 0
```

```
STORE    [array + 0], A
```

```
LOAD     B, [array + 1]
```

```
ADDI     B, 5
```

```
STORE    [array + 1], B
```

```
LOAD     C, [array + 2]
```

```
SUBI     C, 1
```

```
STORE    [array + 2], C
```

```
LOAD     D, [array + 3]
```

```
ADD      D, C
```

```
STORE    [array + 3], D
```

C vs. Assembly

```
// Array
//
// C version

int array[]={1, 2, 3, 4};

int main()
{
    array[0] = 0;

    array[1] += 5;

    array[2] -= 1;

    array[3] += array[2];
}
```

```
; Array
;
; Assembly version

.data
array    BYTE    1, 2, 3, 4

.code

LOADI   A, 0
STORE   [array + 0], A
LOAD    B, [array + 1]
ADDI    B, 5
STORE   [array + 1], B
LOAD    C, [array + 2]
SUBI    C, 1
STORE   [array + 2], C
LOAD    D, [array + 3]
ADD     D, C
STORE   [array + 3], D
```

Machine Code vs. Assembly

```
// Machine Code
```

```
ADDR:   DMEM
0000: 00000001
0001: 00000010
0010: 00000011
0011: 00000100
```

```
IMEM
```

```
0011_00_00_00000000
1010_00_00_00000000
1000_01_00_00000001
0101_01_00_00000101
1010_01_00_00000001
1000_10_00_00000010
0111_10_00_00000001
1010_10_00_00000010
1000_11_00_00000011
0100_11_10_00000000
1010_11_00_00000011
```

```
; Array
```

```
;
```

```
; Assembly version
```

```
.data
```

```
array   BYTE    1, 2, 3, 4
```

```
.code
```

```
LOADI   A, 0
```

```
STORE   [array + 0], A
```

```
LOAD    B, [array + 1]
```

```
ADDI    B, 5
```

```
STORE   [array + 1], B
```

```
LOAD    C, [array + 2]
```

```
SUBI    C, 1
```

```
STORE   [array + 2], C
```

```
LOAD    D, [array + 3]
```

```
ADD     D, C
```

```
STORE   [array + 3], D
```

Machine Code vs. Assembly

```
// Machine Code
```

```
ADDR:   DMEM
```

```
0000: 00000001
```

```
0001: 00000010
```

```
0010: 00000011
```

```
0011: 00000100
```

```
IMEM
```

```
0011_00_00_00000000
```

```
1010_00_00_00000000
```

```
1000_01_00_00000001
```

```
0101_01_00_00000101
```

```
1010_01_00_00000001
```

```
1000_10_00_00000010
```

```
0111_10_00_00000001
```

```
1010_10_00_00000010
```

```
1000_11_00_00000011
```

```
0100_11_10_00000000
```

```
1010_11_00_00000011
```

```
; Array
```

```
;
```

```
; Assembly version
```

```
.data
```

```
array   BYTE    1, 2, 3, 4
```

```
.code
```

```
LOADI   A, 0
```

```
STORE   [array + 0], A
```

```
LOAD    B, [array + 1]
```

```
ADDI    B, 5
```

```
STORE   [array + 1], B
```

```
LOAD    C, [array + 2]
```

```
SUBI    C, 1
```

```
STORE   [array + 2], C
```

```
LOAD    D, [array + 3]
```

```
ADD     D, C
```

```
STORE   [array + 3], D
```

Machine Code vs. Assembly

```
// Machine Code
```

```
ADDR:   DMEM
```

```
0000: 00000001
```

```
0001: 00000010
```

```
0010: 00000011
```

```
0011: 00000100
```

```
IMEM
```

```
0011_00_00_00000000
```

```
1010_00_00_00000000
```

```
1000_01_00_00000001
```

```
0101_01_00_00000101
```

```
1010_01_00_00000001
```

```
1000_10_00_00000010
```

```
0111_10_00_00000001
```

```
1010_10_00_00000010
```

```
1000_11_00_00000011
```

```
0100_11_10_00000000
```

```
1010_11_00_00000011
```

```
; Array
```

```
;
```

```
; Assembly version
```

```
.data
```

```
array   BYTE    1, 2, 3, 4
```

```
.code
```

```
LOADI   A, 0
```

```
STORE   [array + 0], A
```

```
LOAD    B, [array + 1]
```

```
ADDI    B, 5
```

```
STORE   [array + 1], B
```

```
LOAD    C, [array + 2]
```

```
SUBI    C, 1
```

```
STORE   [array + 2], C
```

```
LOAD    D, [array + 3]
```

```
ADD     D, C
```

```
STORE   [array + 3], D
```


Working with Arrays And Looping Over Their Elements

C vs. Assembly

```
// Array plus 5
// C version

int array[]={1, 2, 3, 4};
int N = 4;

int main()
{
    int i;

    // add 5 to all elements
    for( i=0; i< N; i++)
    {
        array[i] += 5;
    }
}
```

```
; Array plus 5
; Assembly version

.data
array    BYTE    1, 2, 3, 4
N        BYTE    4

.code

                LOADI   A, 0                ; i = 0
For:          LOAD    D, [N]                ; D ← N
                CMP     A, D                ; i < N ?
                BRGE   End                  ; if no, exit loop
                LOADF  C, [array + A]      ; load array[i]
                ADDI   C, 5                  ; add 5
                STOREF [array + A], C     ; store the result
Iinc:        ADDI   A, 1                    ; i++
                JUMP   For                  ; next iteration
End:         NOOP

; Register allocation:
; A: i
; B: <not used>
; C: temp variable for loading the array elements
; D: N
```

C vs. Assembly

```
// Array plus 5
// C version

int array[]={1, 2, 3, 4};
int N = 4;

int main()
{
    int i;

    // add 5 to all elements
    for( i=0; i< N; i++)
    {
        array[i] += 5;
    }
}
```

```
; Array plus 5
; Assembly version

.data
array    BYTE    1, 2, 3, 4
N        BYTE    4

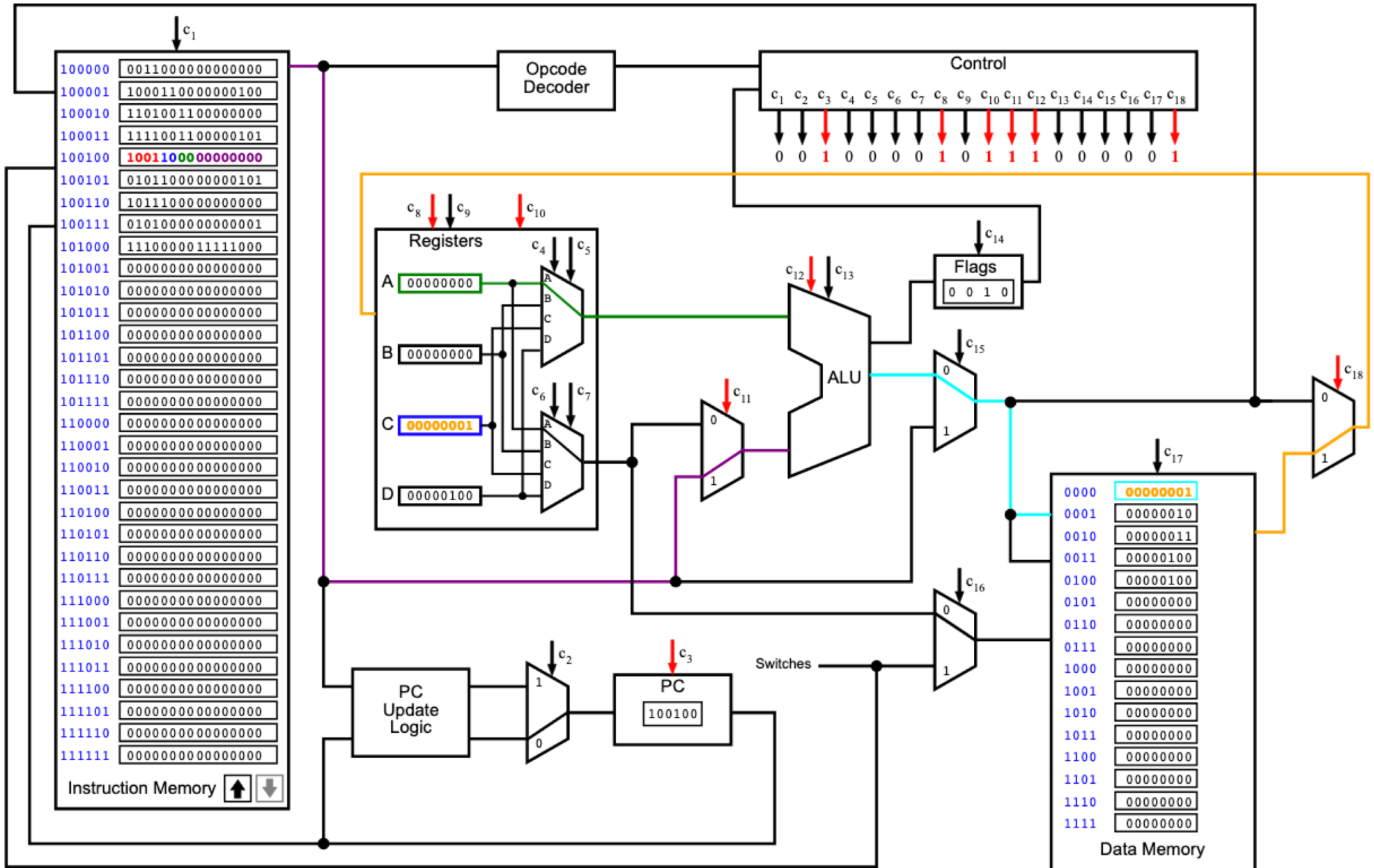
.code

                LOADI   A, 0                ; i = 0
For:          LOAD    D, [N]                ; D ← N
                CMP     A, D                ; i < N ?
                BRGE   End                  ; if no, exit loop
                LOADF  C, [array + A]       ; load array[i]
                ADDI   C, 5                  ; add 5
                STOREF [array + A], C      ; store the result
Iinc:        ADDI   A, 1                    ; i++
                JUMP   For                  ; next iteration
End:         NOOP

; Register allocation:
; A: i
; B: <not used>
; C: temp variable for loading the array elements
; D: N
```

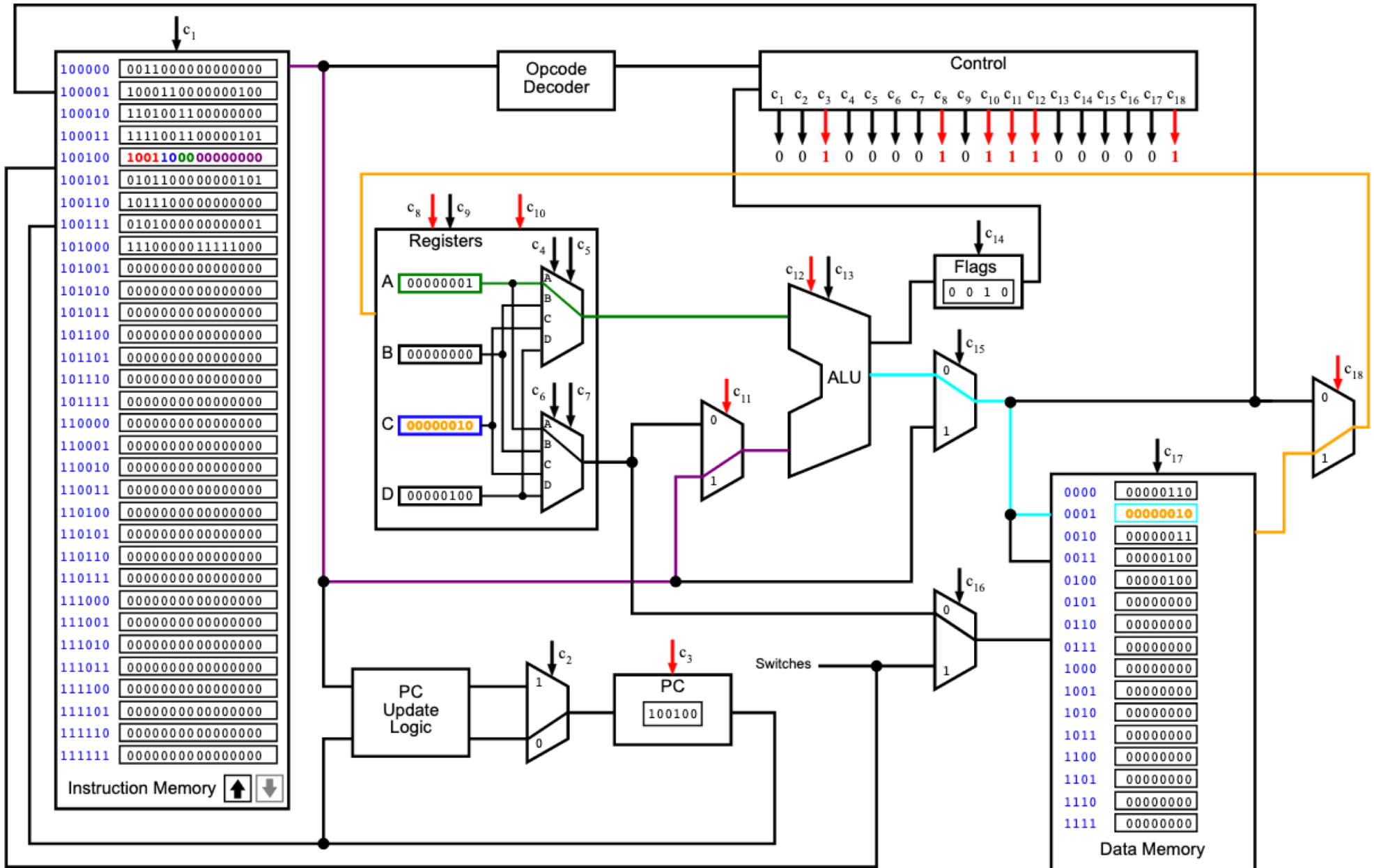
Current Instruction: LOADF C, [array+A]

i281 CPU Running: ArrayPlusFive



Current Instruction: LOADF C, [array+A]

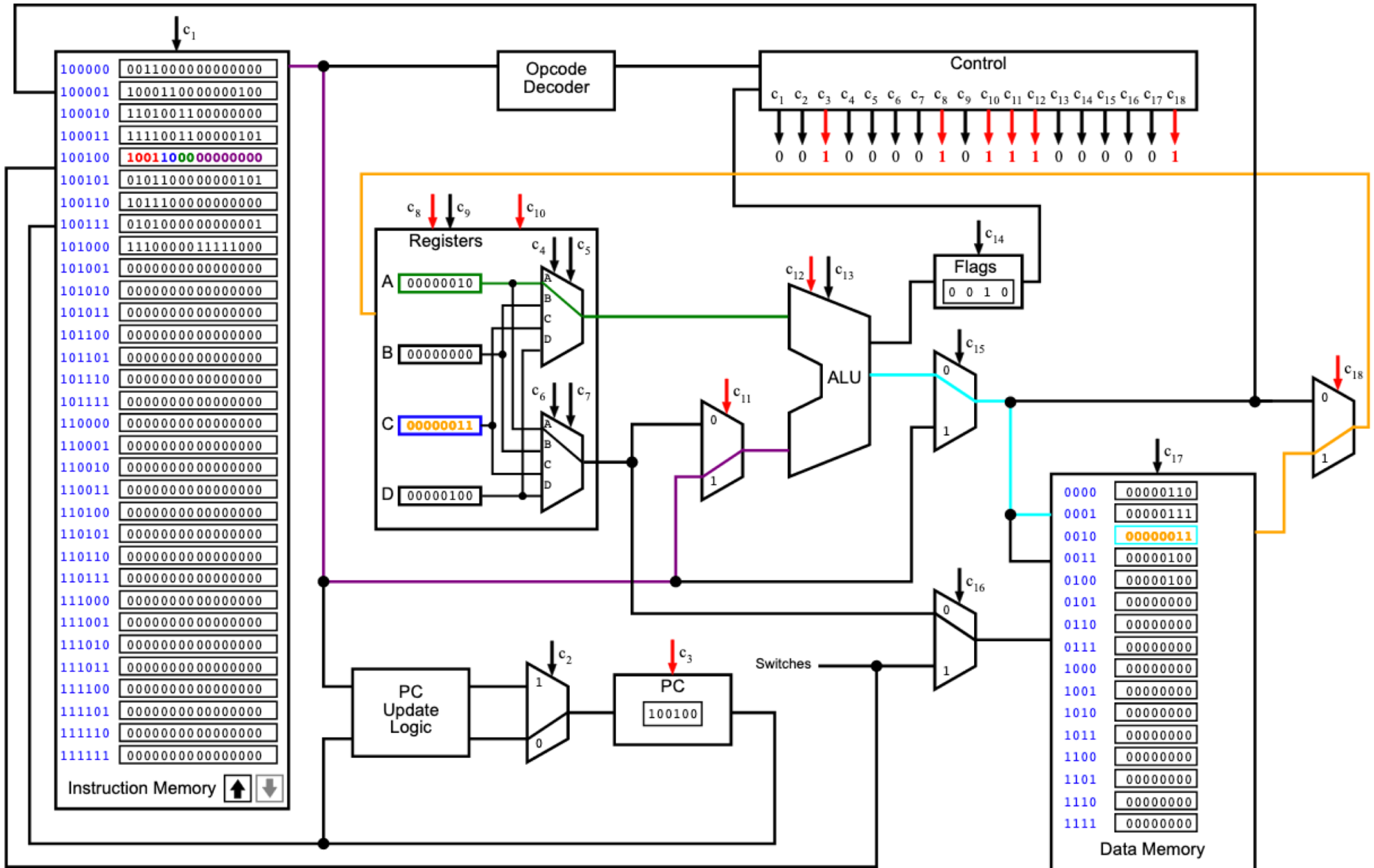
i281 CPU Running: ArrayPlusFive



2nd iteration

Current Instruction: LOADF C, [array+A]

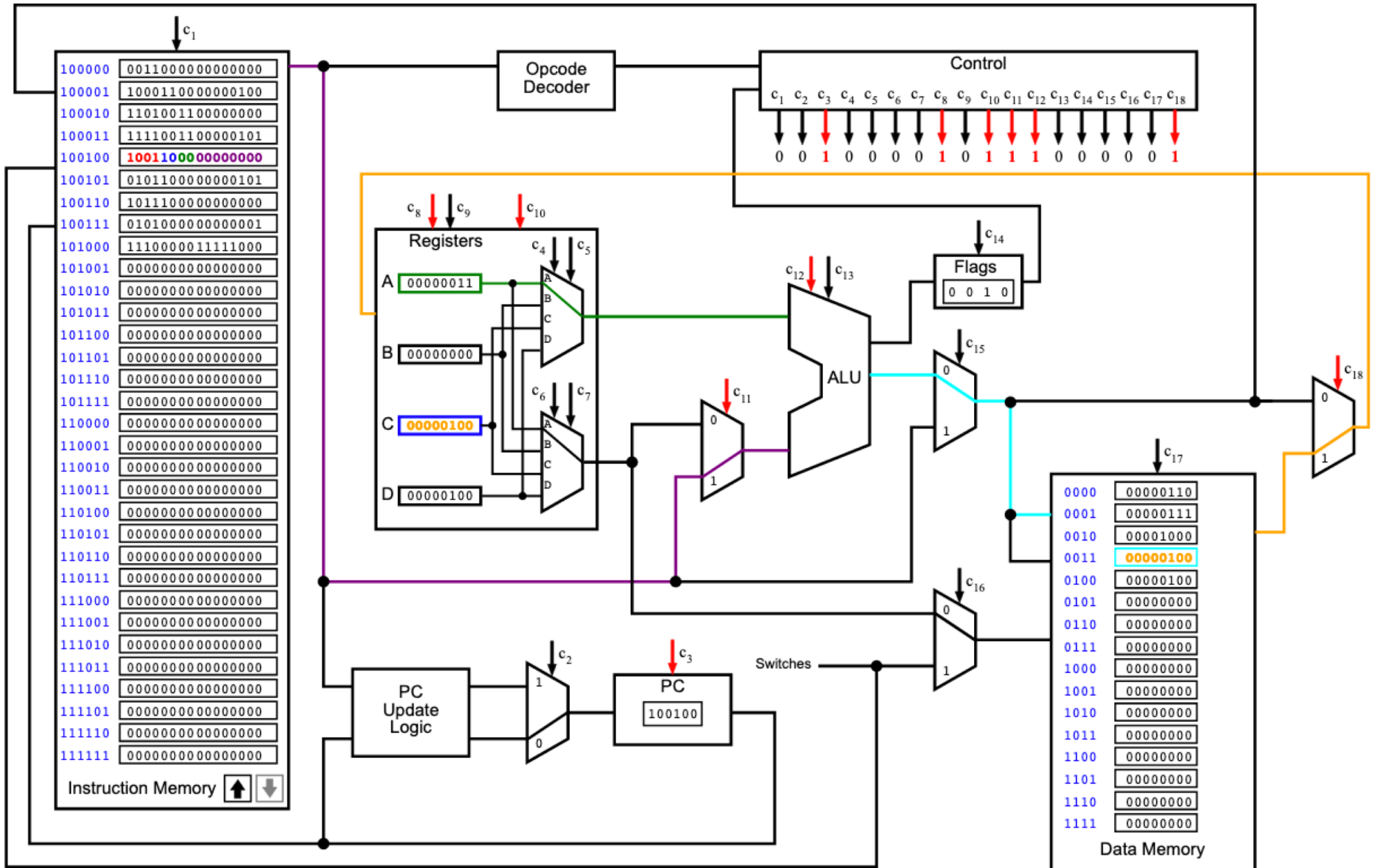
i281 CPU Running: ArrayPlusFive



3rd iteration

Current Instruction: LOADF C, [array+A]

i281 CPU Running: ArrayPlusFive



4th iteration

C vs. Assembly

```
// Array plus 5
// C version

int array[]={1, 2, 3, 4};
int N = 4;

int main()
{
    int i;

    // add 5 to all elements
    for( i=0; i< N; i++)
    {
        array[i] += 5;
    }
}
```

```
; Array plus 5
; Assembly version

.data
array    BYTE    1, 2, 3, 4
N        BYTE    4

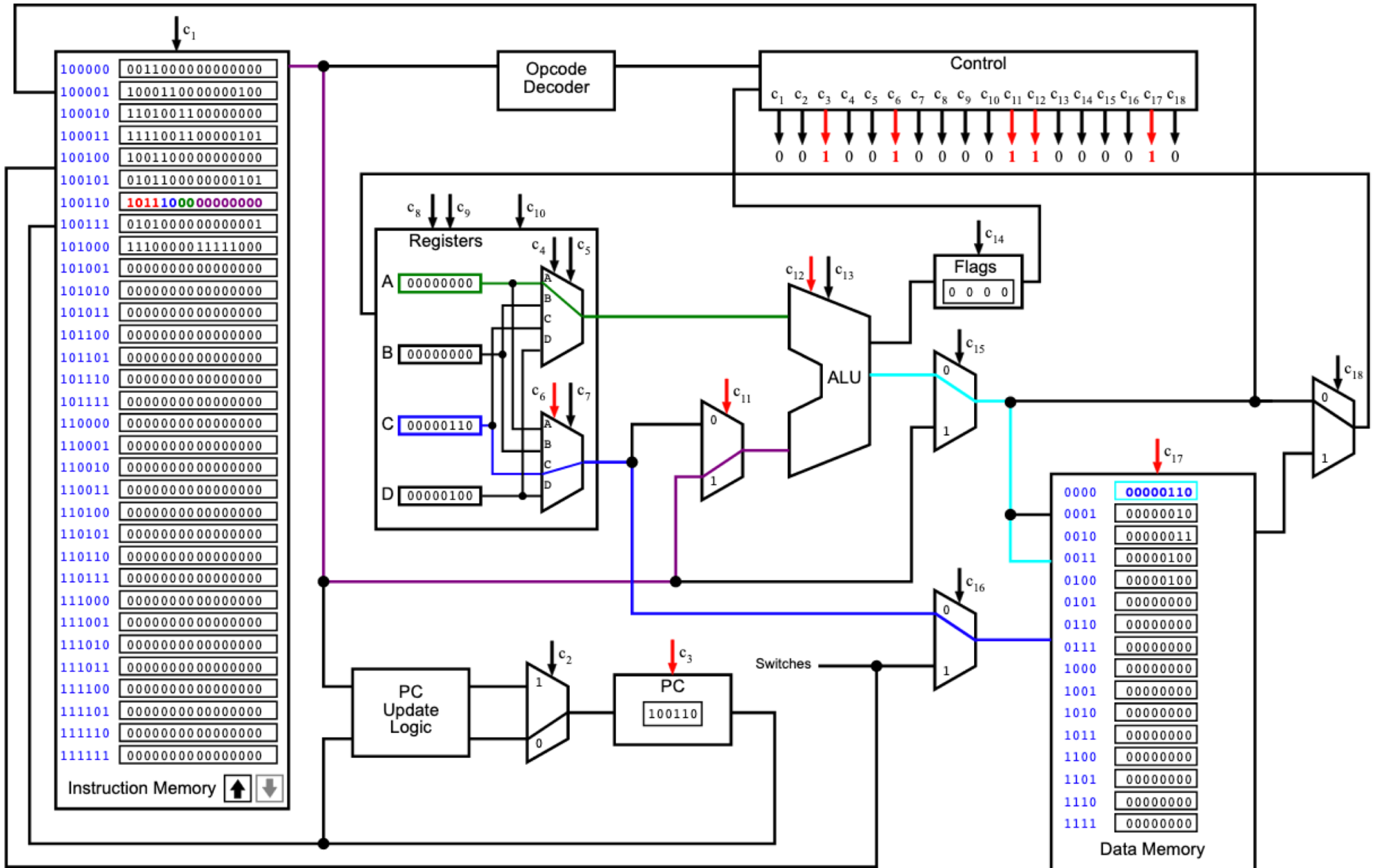
.code

                LOADI   A, 0                ; i = 0
For:          LOAD    D, [N]                ; D ← N
                CMP     A, D                ; i < N ?
                BRGE   End                  ; if no, exit loop
                LOADF  C, [array + A]       ; load array[i]
                ADDI   C, 5                 ; add 5
                STOREF [array + A], C       ; store the result
Iinc:        ADDI   A, 1                    ; i++
                JUMP   For                  ; next iteration
End:         NOOP

; Register allocation:
; A: i
; B: <not used>
; C: temp variable for loading the array elements
; D: N
```


Current Instruction: STOREF [array+A], C

i281 CPU Running: ArrayPlusFive



C vs. Assembly

```
// Array plus 5
// C version

int array[]={1, 2, 3, 4};
int N = 4;

int main()
{
    int i;

    // add 5 to all elements
    for( i=0; i< N; i++)
    {
        array[i] += 5;
    }
}
```

```
; Array plus 5
; Assembly version

.data
array    BYTE    1, 2, 3, 4
N        BYTE    4

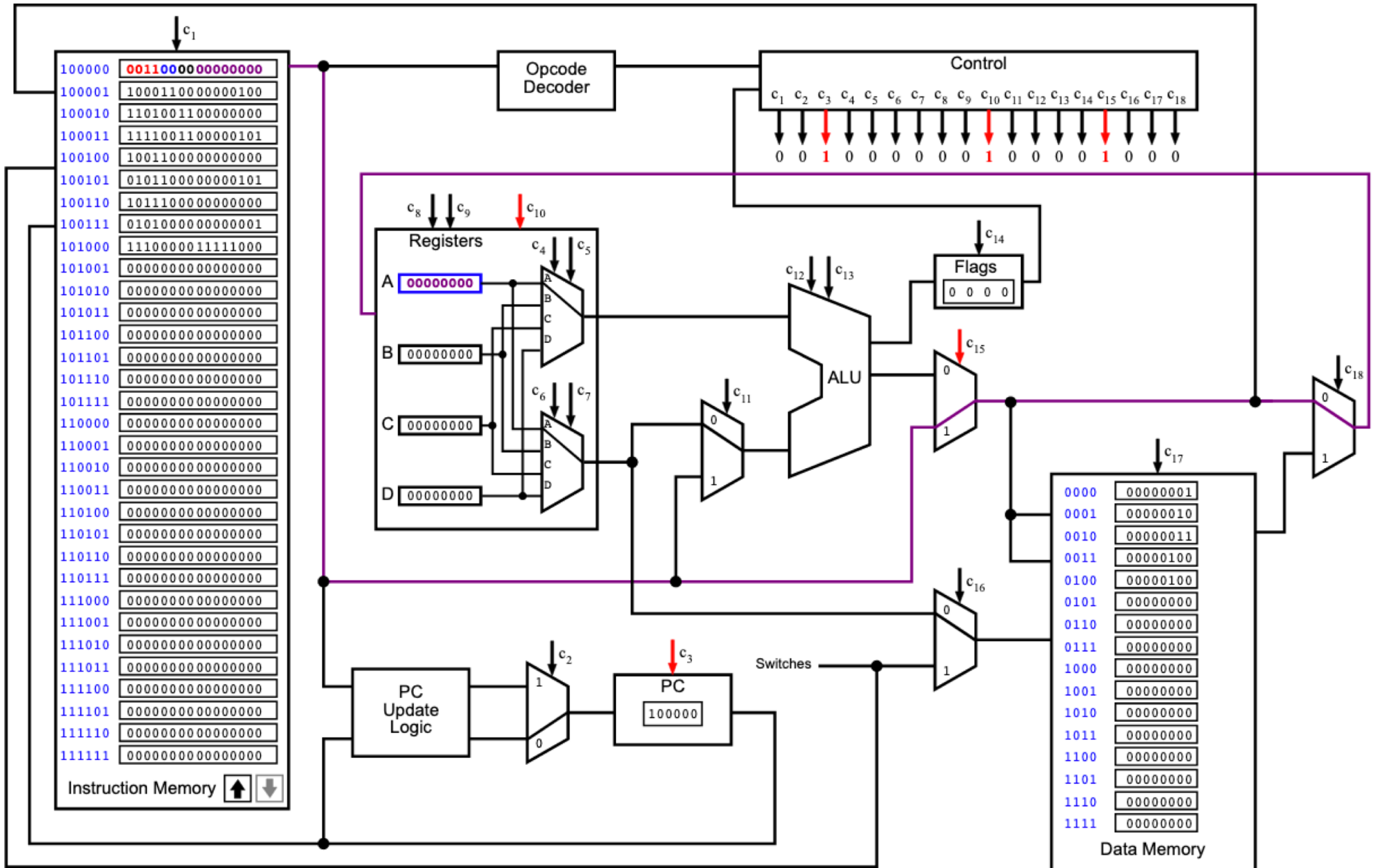
.code

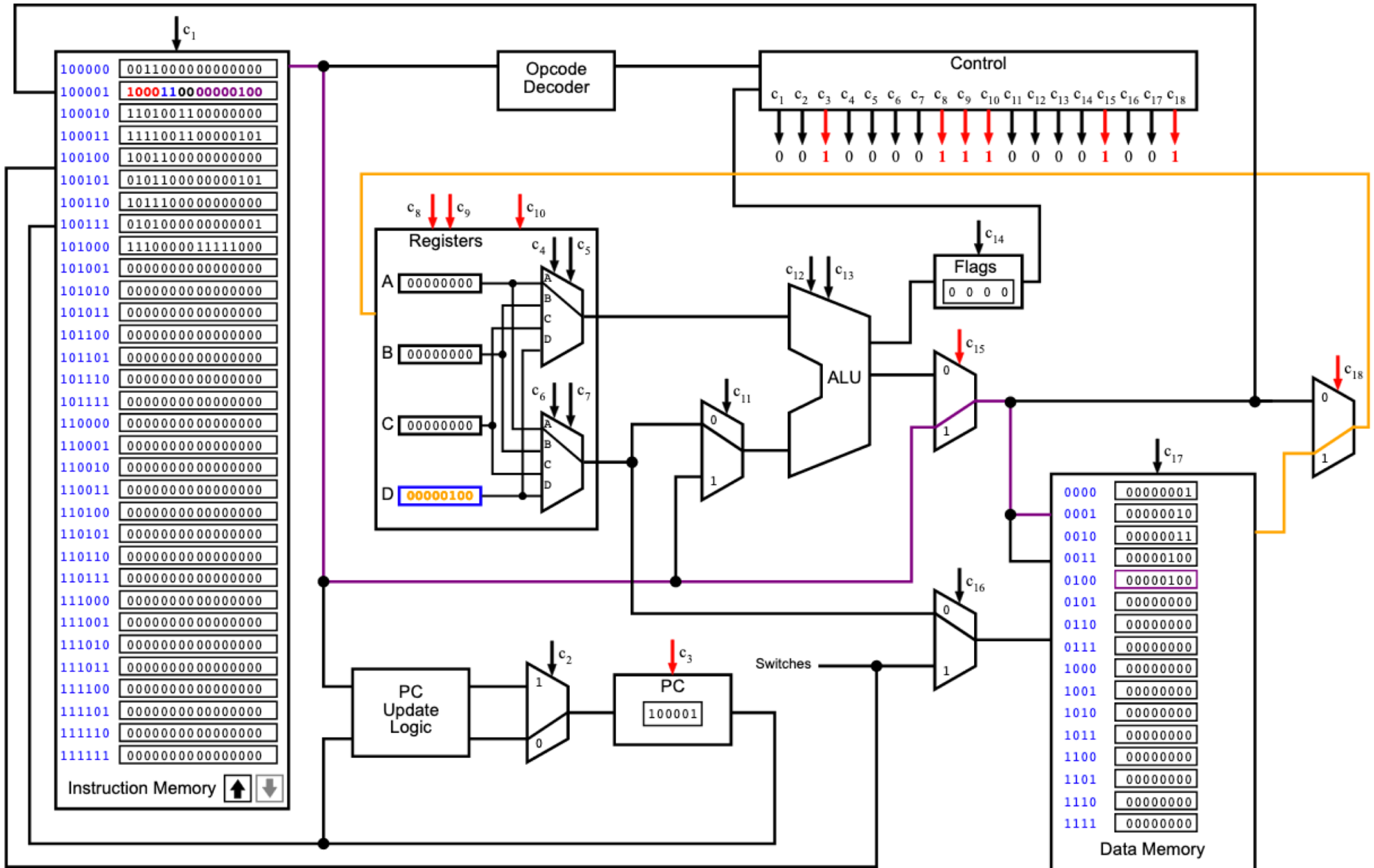
                LOADI   A, 0                ; i = 0
For:          LOAD    D, [N]                ; D ← N
                CMP     A, D                ; i < N ?
                BRGE   End                  ; if no, exit loop
                LOADF  C, [array + A]      ; load array[i]
                ADDI   C, 5                 ; add 5
                STOREF [array + A], C     ; store the result
Iinc:        ADDI   A, 1                    ; i++
                JUMP   For                  ; next iteration
End:         NOOP

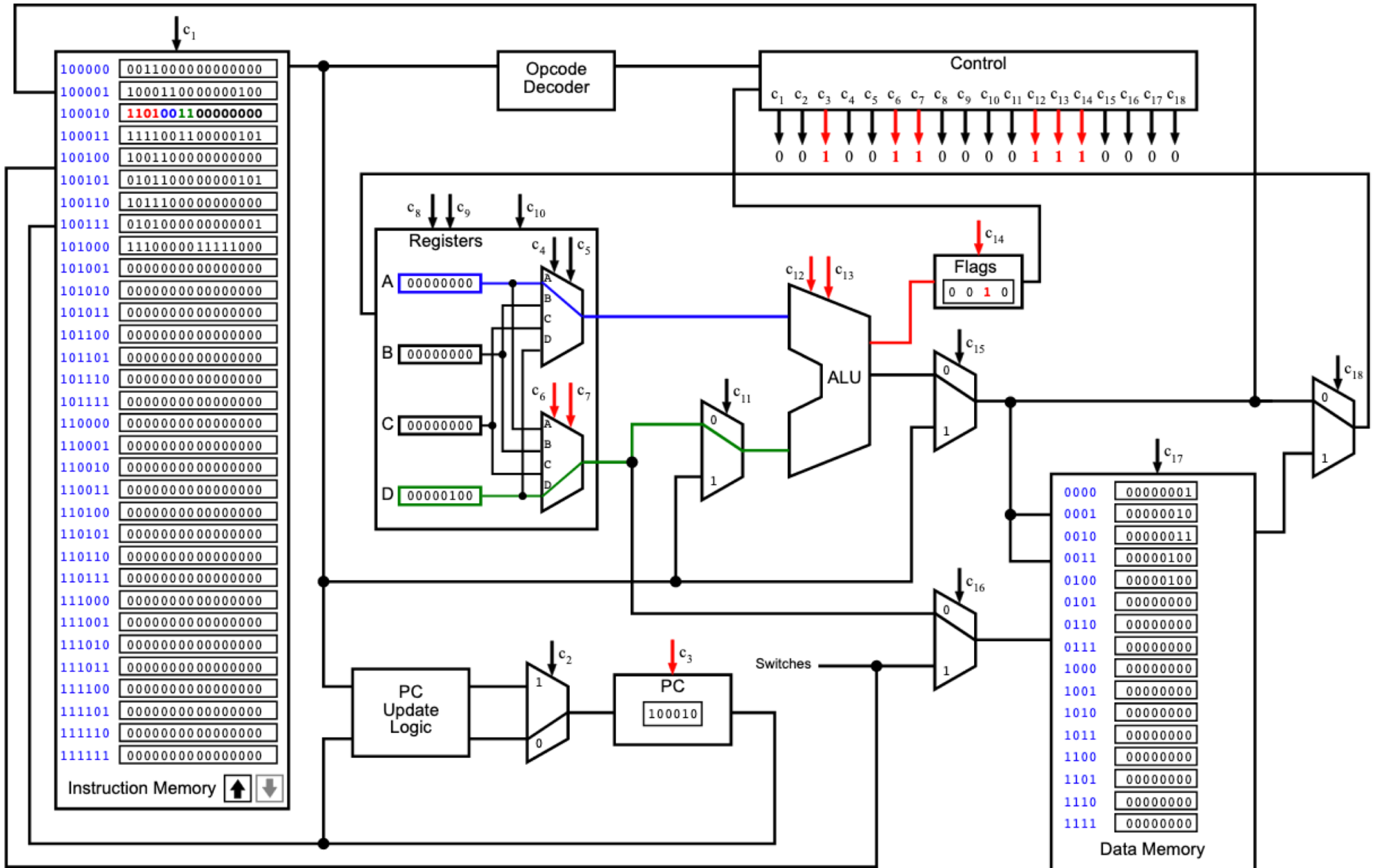
; Register allocation:
; A: i
; B: <not used>
; C: temp variable for loading the array elements
; D: N
```

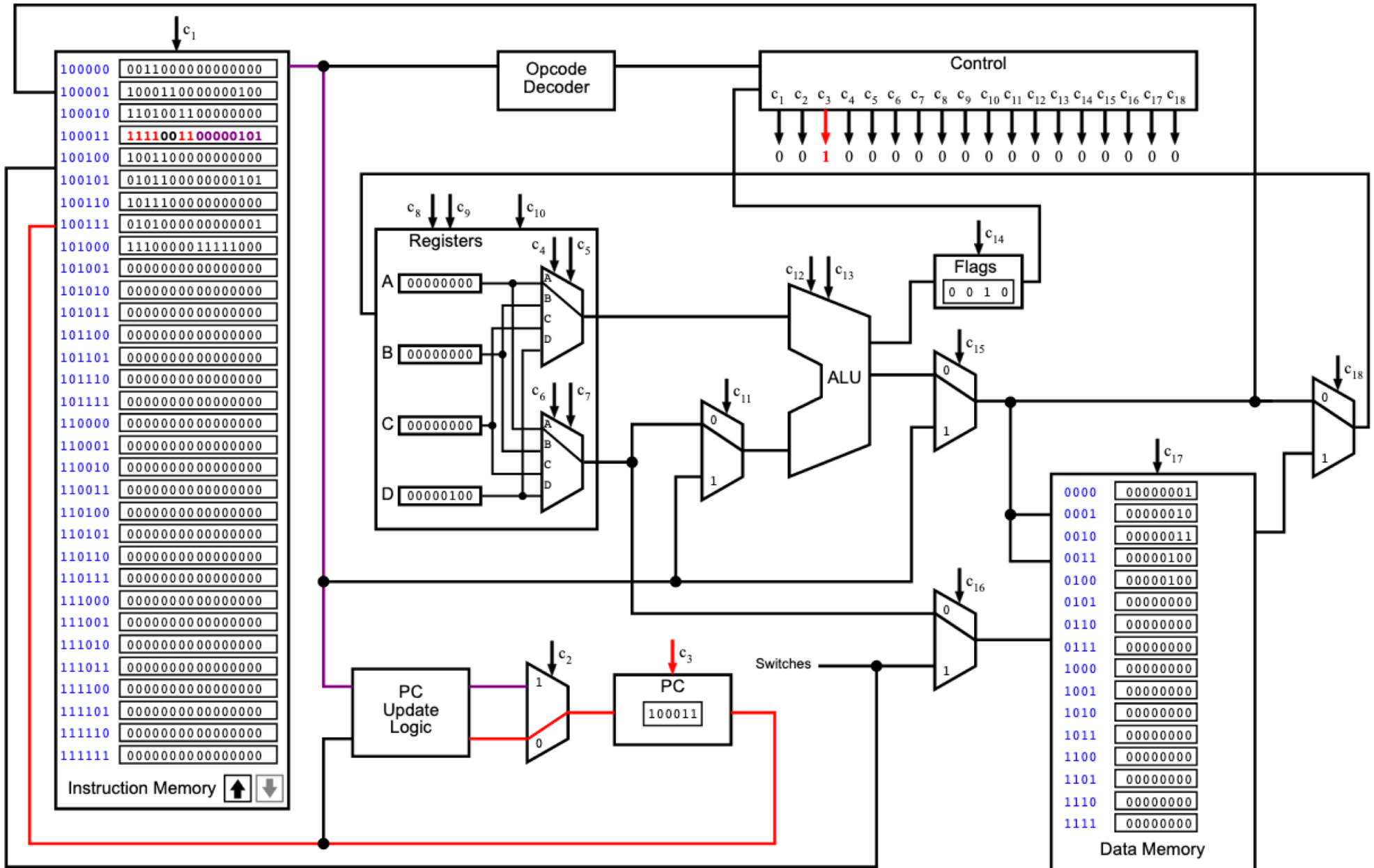
Current Instruction: **LOADI A, 0**

i281 CPU Running: **ArrayPlusFive**



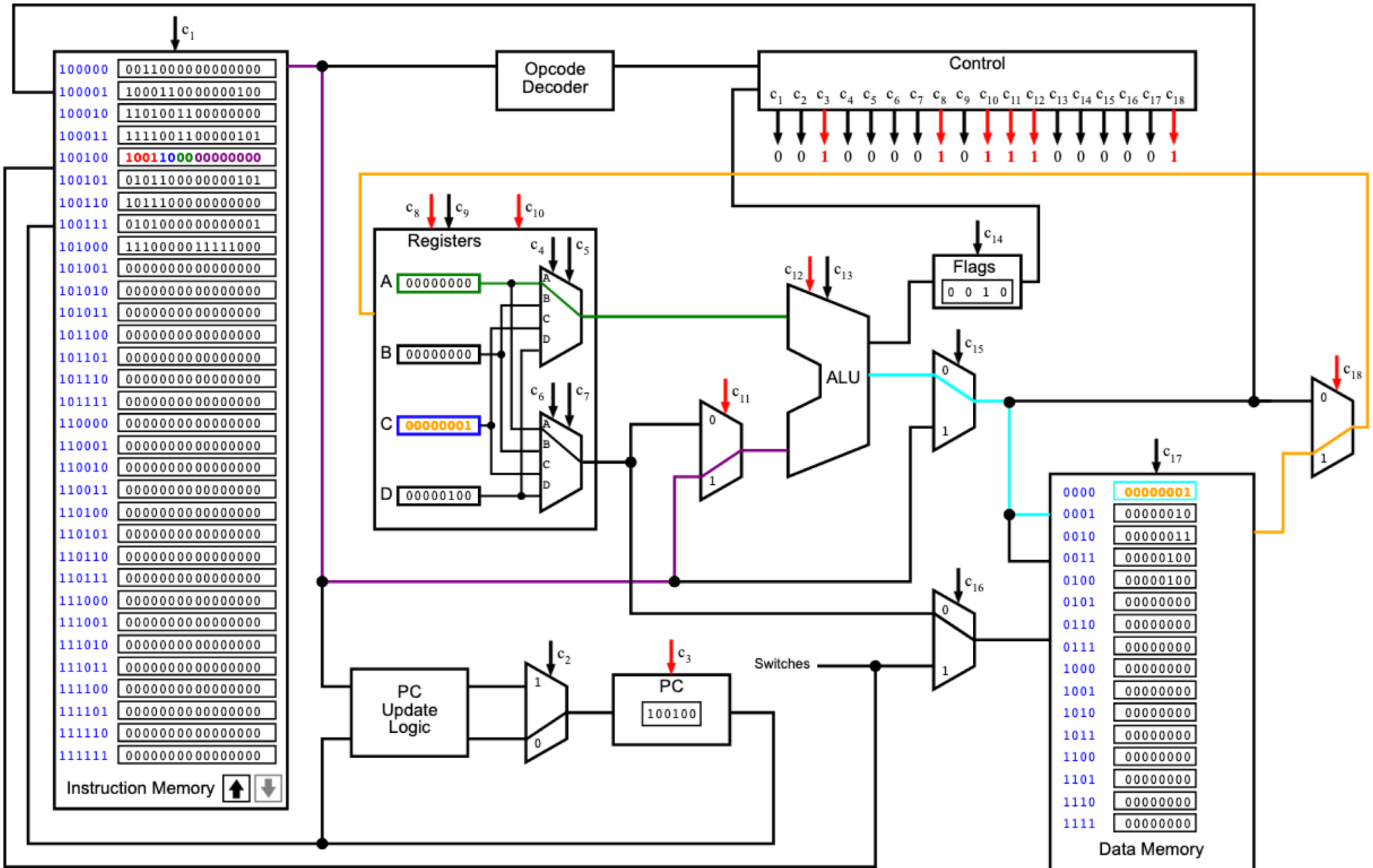


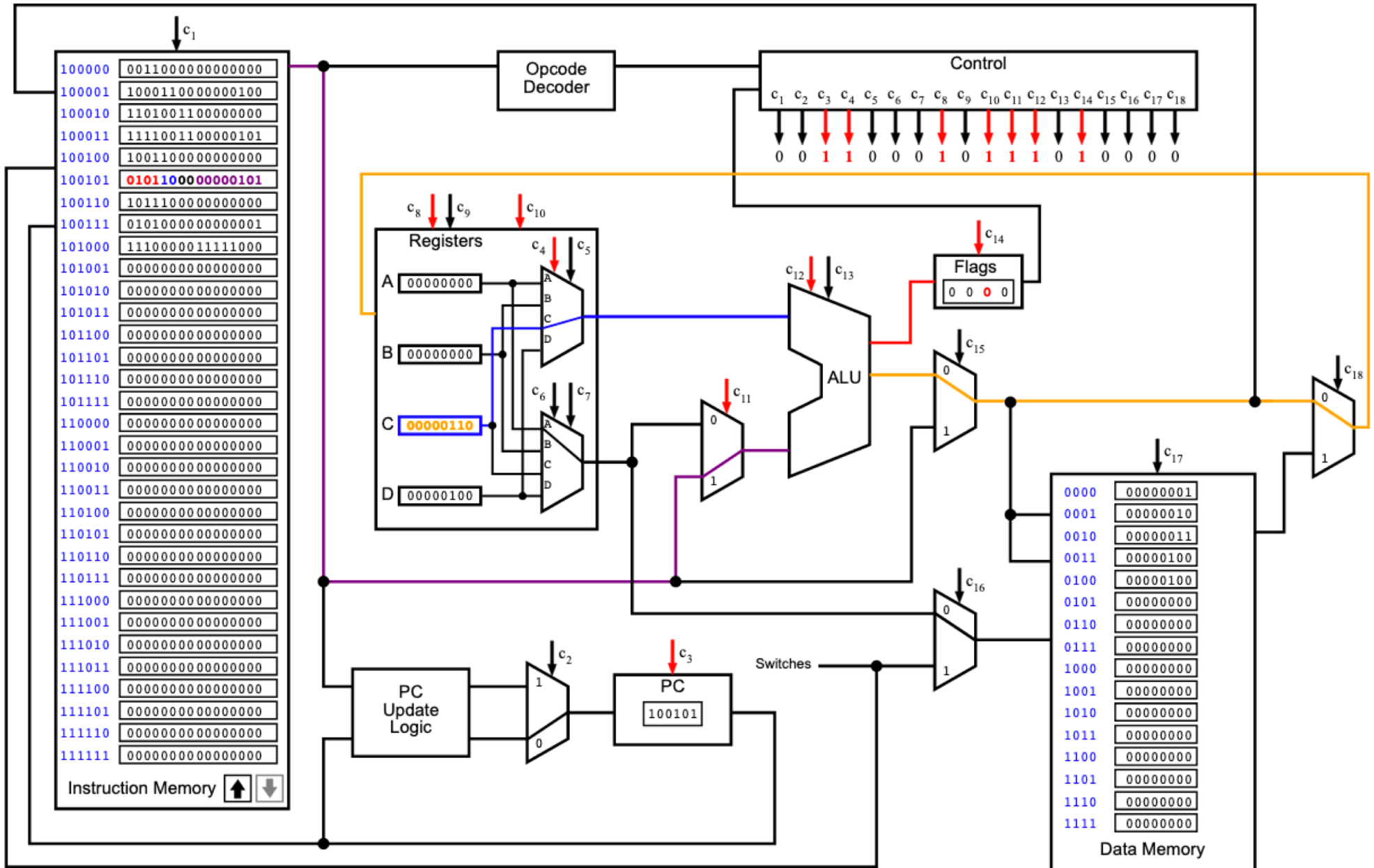




Current Instruction: LOADF C, [array+A]

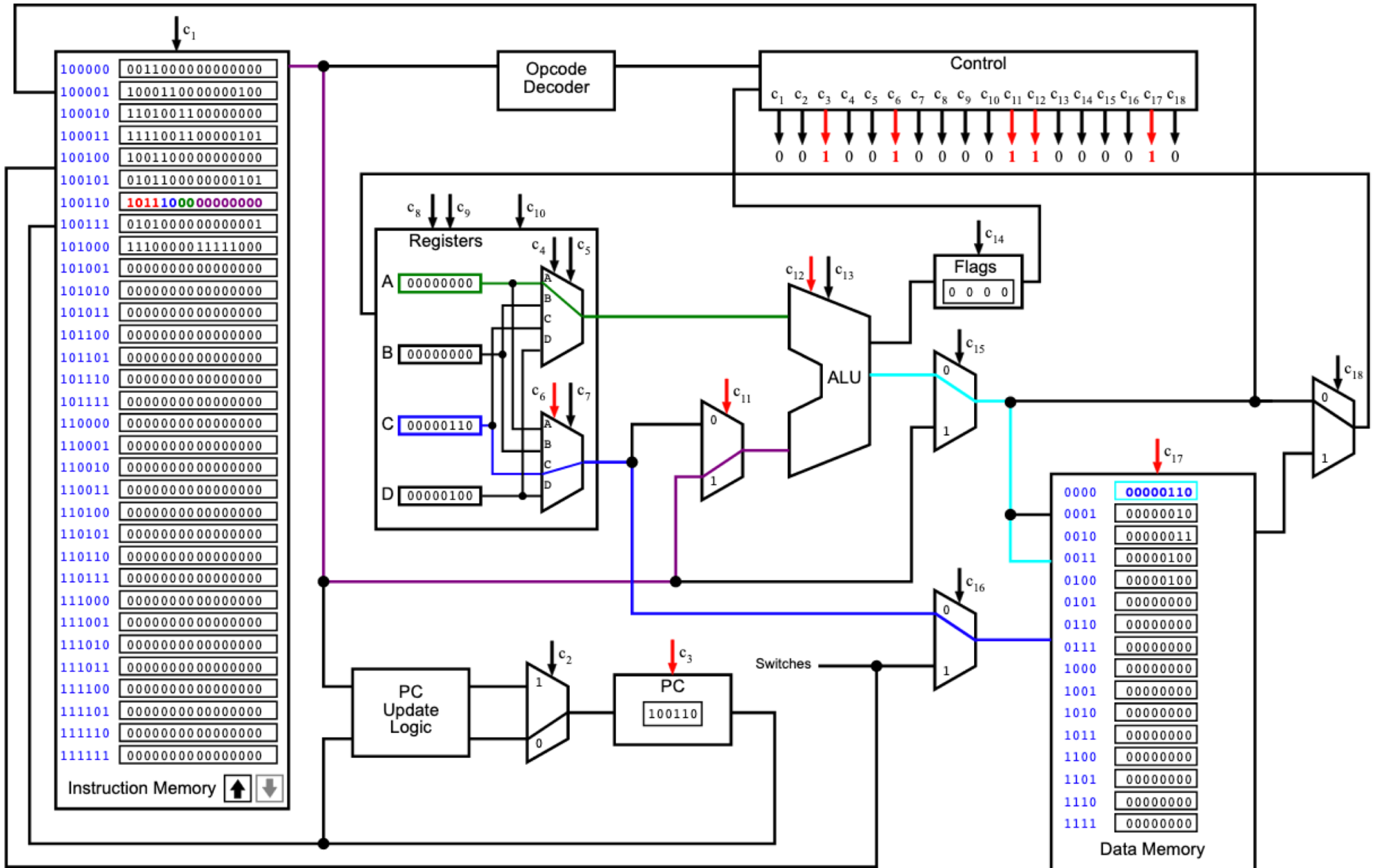
i281 CPU Running: ArrayPlusFive





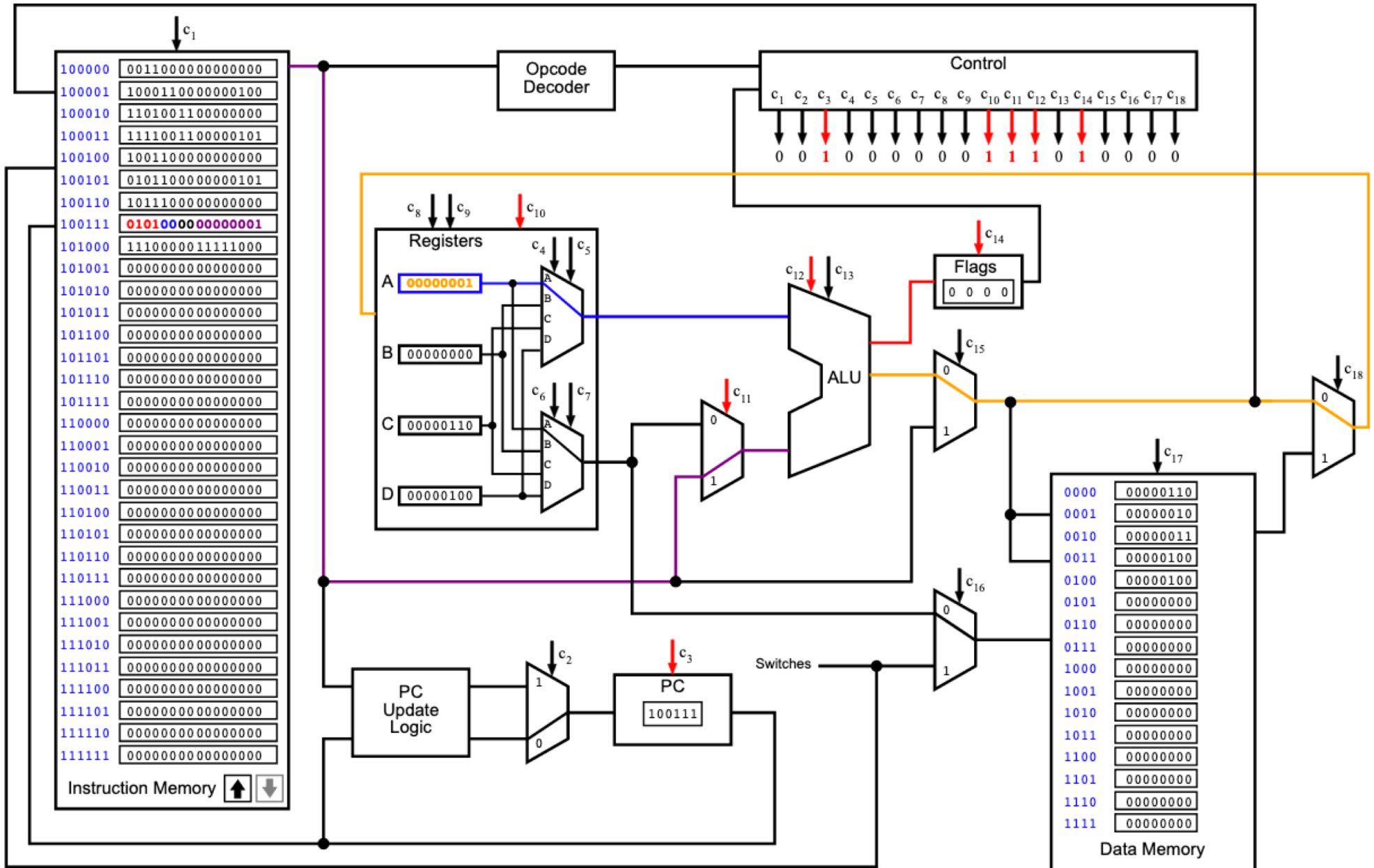
Current Instruction: STOREF [array+A], C

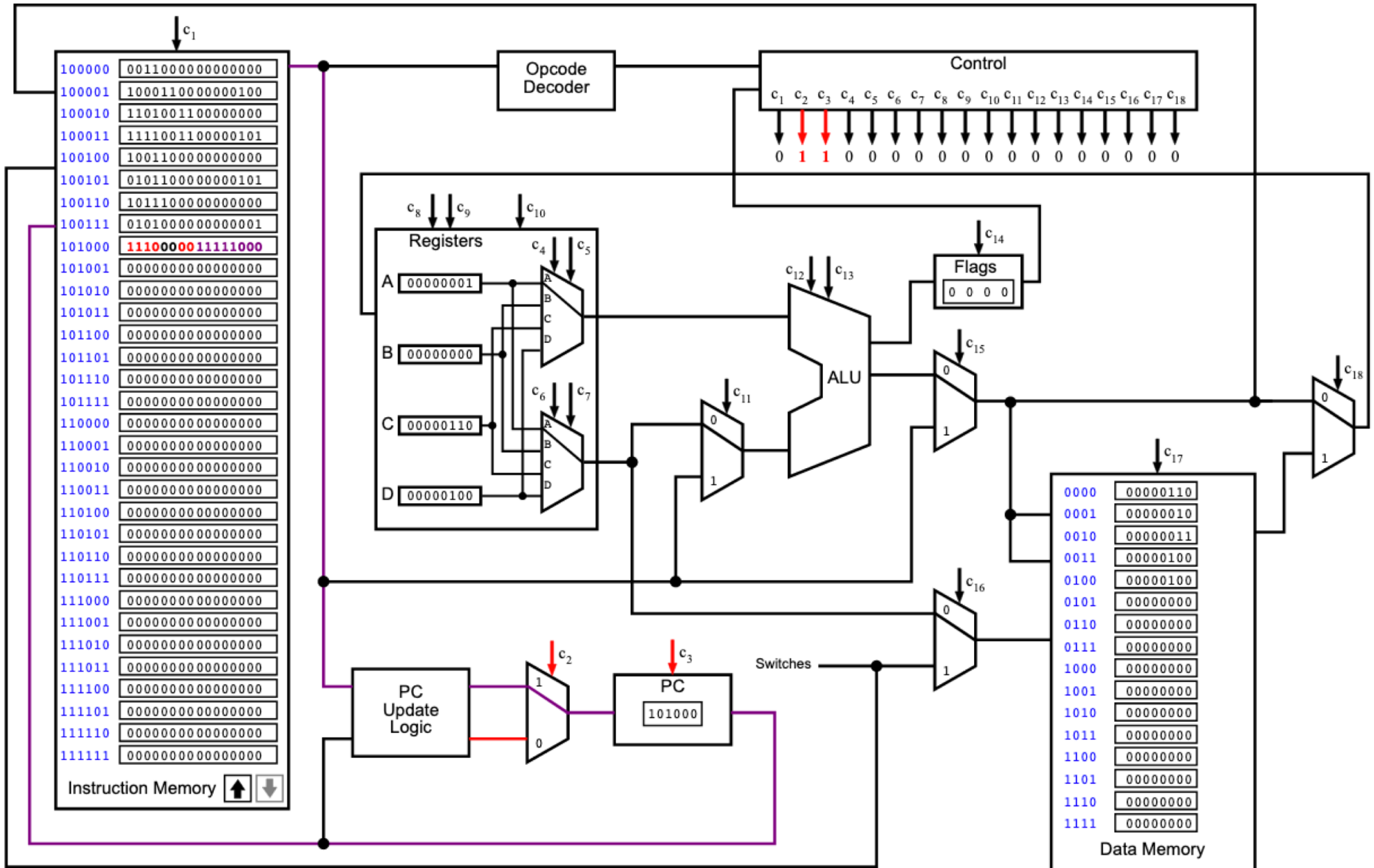
i281 CPU Running: ArrayPlusFive

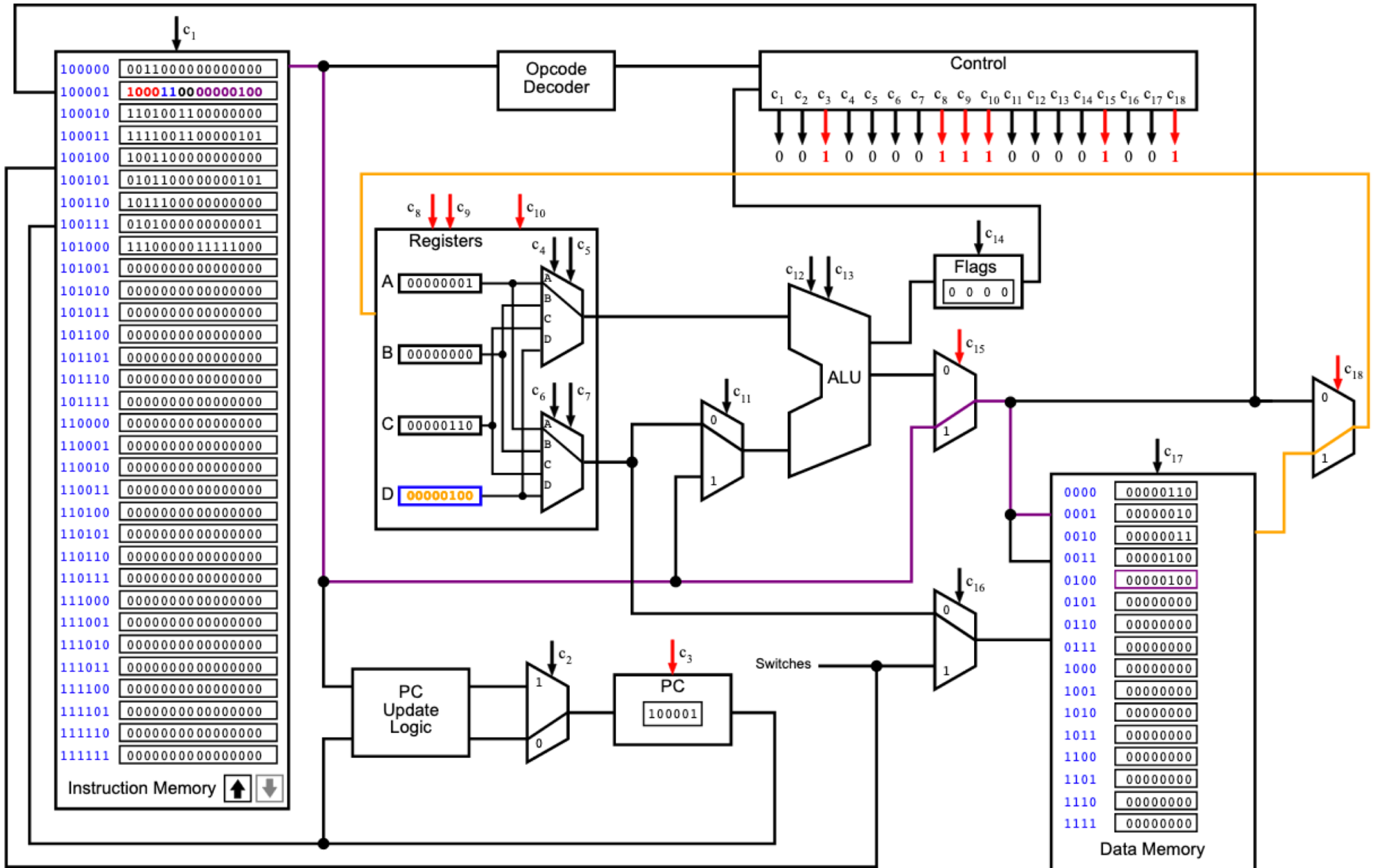


Current Instruction: ADDI A, 1

i281 CPU Running: ArrayPlusFive







**For Loop Example:
Add the numbers from 1 to 5**

**For Loop Example:
Add the numbers from 1 to 5**

C Language v.s. Assembly Language

C Version

```
// C Version
//
// Add the numbers from 1 to 5 using a for loop.

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++)
        sum+=i;

    // printf("%d\n", sum);
}
```

i281 Assembly Version

.data

```
N      BYTE    5
i      BYTE    ?
sum    BYTE    ?
```

.code

```
      LOADI   B, 0      ; sum=0
      LOADI   A, 1      ; i=1
      LOAD    D, [N]    ; register_D=N
Loop:  CMP     A, D      ; i<=N ?
      BRG     End       ; exit if i>N
Add:   ADD    B, A      ; sum+=i
      ADDI   A, 1      ; i++
      JUMP   Loop      ; next iteration
End:   STORE  [sum], B  ; update the memory for sum
```

; Register allocation:

; A: i

; B: sum

; C: <not used>

; D: N

i281 Assembly Version

.data

```
N      BYTE    5
i      BYTE    ?
sum    BYTE    ?
```

.code

```
      LOADI   B, 0      ; sum=0
      LOADI   A, 1      ; i=1
      LOAD    D, [N]    ; register_D=N
Loop:  CMP     A, D      ; i<=N ?
      BRG     End      ; exit if i>N
Add:   ADD    B, A      ; sum+=i
      ADDI   A, 1      ; i++
      JUMP   Loop      ; next iteration
End:   STORE  [sum], B  ; update the memory for sum
```

; Register allocation:

; A: i

; B: sum

; C: <not used>

; D: N

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD    B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N          BYTE    5
i          BYTE    ?
sum        BYTE    ?

.code

        LOADI   B, 0          ; sum=0
        LOADI   A, 1          ; i=1
        LOAD    D, [N]        ; register_D=N
Loop:   CMP     A, D          ; i<=N ?
        BRG     End          ; exit if i>N
Add:    ADD    B, A          ; sum+=i
        ADDI   A, 1          ; i++
        JUMP   Loop          ; next iteration
End:    STORE  [sum], B      ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD    B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop       ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

i=1

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI  A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

This has no analog in the C version,
which is written in a high-level language.

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Load the value of N into register D.

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop       ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

i=2

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD    B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop       ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop       ; next iteration
End:    STORE   [sum], B    ; write B to sum
```


Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

i=3

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

i=4

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG    End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]     ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG    End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop       ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop       ; next iteration
End:    STORE   [sum], B    ; write B to sum
```


Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

i=5

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD    B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop       ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop       ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

i=6

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

For Loop Example:
Add the numbers from 1 to 5

Assembly Language v.s. Machine Language

i281 Assembly Code

.data

```
N          BYTE    5  
i          BYTE    ?  
sum       BYTE    ?
```

.code

```
          LOADI  B, 0          ; sum=0  
          LOADI  A, 1          ; i=1  
          LOAD   D, [N]        ; register_D=N  
Loop:    CMP    A, D          ; i<=N ?  
          BRG    End          ; exit if i>N  
Add:    ADD    B, A          ; sum+=i  
          ADDI   A, 1          ; i++  
          JUMP   Loop         ; next iteration  
End:    STORE  [sum], B     ; update the memory for sum
```


i281 Assembly Code

.data

```
N          BYTE    5  
i          BYTE    ?  
sum       BYTE    ?
```

.code

```
          LOADI  B, 0  
          LOADI  A, 1  
          LOAD   D, [N]  
Loop:    CMP    A, D  
          BRG    End  
Add:    ADD    B, A  
          ADDI   A, 1  
          JUMP   Loop  
End:    STORE  [sum], B
```

Mapping Assembly to Machine Code

.data

```
N      BYTE  5
i      BYTE  ?
sum    BYTE  ?
```

Data Memory:

```
00000101
00000000
00000000
```

.code

```
      LOADI  B, 0
      LOADI  A, 1
      LOAD   D, [N]
Loop:  CMP    A, D
      BRG    End
Add:   ADD   B, A
      ADDI  A, 1
      JUMP  Loop
End:   STORE [sum], B
```

Code Memory:

```
0011010000000000
0011000000000001
1000110000000000
1101001100000000
1111001000000011
0100010000000000
0101000000000001
1110000011111011
1010010000000010
```

Assembly Language

Machine Language

Mapping Assembly to Machine Code

.data

N **BYTE** **5**
i **BYTE** **?**
sum **BYTE** **?**

.code

LOADI **B, 0**
 LOADI **A, 1**
 LOAD **D, [N]**
Loop: **CMP** **A, D**
 BRG **End**
Add: **ADD** **B, A**
 ADDI **A, 1**
 JUMP **Loop**
End: **STORE** **[sum], B**

Data Memory:

00000101
00000000
00000000

Code Memory:

00110100_00000000
00110000_00000001
10001100_00000000
11010011_00000000
11110010_00000011
01000100_00000000
01010000_00000001
11100000_11111011
10100100_00000010

Assembly Language

Machine Language

Mapping Assembly to Machine Code

.data

N **BYTE** **5**
i **BYTE** **?**
sum **BYTE** **?**

.code

LOADI **B, 0**
 LOADI **A, 1**
 LOAD **D, [N]**
Loop: **CMP** **A, D**
 BRG **End**
Add: **ADD** **B, A**
 ADDI **A, 1**
 JUMP **Loop**
End: **STORE** **[sum], B**

Data Memory:

00000101
00000000
00000000

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

Assembly Language

Machine Language

Mapping Assembly to Machine Code

.data

```
N      BYTE  5
i      BYTE  ?
sum    BYTE  ?
```

Data Memory:

```
00000101
00000000
00000000
```

.code

```
      LOADI  B, 0
      LOADI  A, 1
      LOAD   D, [N]
Loop:  CMP    A, D
      BRG    End
Add:   ADD   B, A
      ADDI  A, 1
      JUMP  Loop
End:   STORE [sum], B
```

Code Memory:

```
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

Mapping Assembly to Machine Code

.data

```
N      BYTE  5
i      BYTE  ?
sum    BYTE  ?
```

Data Memory:

```
00000101
00000000
00000000
```

.code

```
      LOADI  B, 0
      LOADI  A, 1
      LOAD   D, [N]
Loop:  CMP    A, D
      BRG    End
Add:   ADD   B, A
      ADDI  A, 1
      JUMP  Loop
End:   STORE [sum], B
```

Code Memory:

```
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

Mapping Assembly to Machine Code

.data

```
N      BYTE  5
i      BYTE  ?
sum    BYTE  ?
```

Data Memory:

```
00000101
00000000
00000000
```

.code

```
      LOADI  B, 0
      LOADI  A, 1
      LOAD   D, [N]
Loop:  CMP    A, D
      BRG    End
Add:   ADD   B, A
      ADDI  A, 1
      JUMP  Loop
End:   STORE [sum], B
```

Code Memory:

```
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

Mapping Assembly to Machine Code

.data

```
N      BYTE  5
i      BYTE  ?
sum    BYTE  ?
```

Data Memory:

```
00000101
00000000
00000000
```

.code

```
      LOADI  B, 0
      LOADI  A, 1
      LOAD   D, [N]
Loop:  CMP    A, D
      BRG    End
Add:   ADD   B, A
      ADDI  A, 1
      JUMP  Loop
End:   STORE [sum], B
```

Code Memory:

```
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```


OPCODE Mapping

.data

N **BYTE** **5**
i **BYTE** **?**
sum **BYTE** **?**

Data Memory:

00000101
00000000
00000000

.code

LOADI **B, 0**
 LOADI **A, 1**
 LOAD **D, [N]**
Loop: **CMP** **A, D**
 BRG **End**
Add: **ADD** **B, A**
 ADDI **A, 1**
 JUMP **Loop**
End: **STORE** **[sum], B**

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

OPCODE Mapping

.data

N **BYTE** **5**
i **BYTE** **?**
sum **BYTE** **?**

Data Memory:

00000101
00000000
00000000

.code

LOADI **B, 0**
 LOADI **A, 1**
 LOAD **D, [N]**
Loop: **CMP** **A, D**
 BRG **End**
Add: **ADD** **B, A**
 ADDI **A, 1**
 JUMP **Loop**
End: **STORE** **[sum], B**

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

Register Parameter Mapping

.data

N	BYTE	5
i	BYTE	?
sum	BYTE	?

Data Memory:

00000101
00000000
00000000

.code

	LOADI	B, 0
	LOADI	A, 1
	LOAD	D, [N]
Loop:	CMP	A, D
	BRG	End
Add:	ADD	B, A
	ADDI	A, 1
	JUMP	Loop
End:	STORE	[sum], B

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

Register Parameter Mapping

.data

N	BYTE	5
i	BYTE	?
sum	BYTE	?

Data Memory:

00000101
00000000
00000000

.code

	LOADI	B, 0
	LOADI	A, 1
	LOAD	D, [N]
Loop:	CMP	A, D
	BRG	End
Add:	ADD	B, A
	ADDI	A, 1
	JUMP	Loop
End:	STORE	[sum], B

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

Second Register Parameter Mapping

.data

N **BYTE** **5**
i **BYTE** **?**
sum **BYTE** **?**

Data Memory:

00000101
00000000
00000000

.code

LOADI **B**, **0**
 LOADI **A**, **1**
 LOAD **D**, **[N]**
Loop: **CMP** **A**, **D**
 BRG **End**
Add: **ADD** **B**, **A**
 ADDI **A**, **1**
 JUMP **Loop**
End: **STORE** **[sum]**, **B**

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

Second Register Parameter Mapping

.data

N	BYTE	5
i	BYTE	?
sum	BYTE	?

Data Memory:

00000101
00000000
00000000

.code

	LOADI	B, 0
	LOADI	A, 1
	LOAD	D, [N]
Loop:	CMP	A, D
	BRG	End
Add:	ADD	B, A
	ADDI	A, 1
	JUMP	Loop
End:	STORE	[sum], B

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

Value / Address / Offset Mapping

.data

N	BYTE	5
i	BYTE	?
sum	BYTE	?

Data Memory:

00000101
00000000
00000000

.code

	LOADI	B, 0
	LOADI	A, 1
	LOAD	D, [N]
Loop:	CMP	A, D
	BRG	End
Add:	ADD	B, A
	ADDI	A, 1
	JUMP	Loop
End:	STORE	[sum], B

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

Value / Address / Offset Mapping

.data

N	BYTE	5
i	BYTE	?
sum	BYTE	?

Data Memory:

00000101
00000000
00000000

.code

	LOADI	B, 0
	LOADI	A, 1
	LOAD	D, [N]
Loop:	CMP	A, D
	BRG	End
Add:	ADD	B, A
	ADDI	A, 1
	JUMP	Loop
End:	STORE	[sum], B

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

“Don’t care” bits ...

.data

```
N      BYTE    5
i      BYTE    ?
sum    BYTE    ?
```

Data Memory:

```
00000101
00000000
00000000
```

.code

```
      LOADI   B, 0
      LOADI   A, 1
      LOAD    D, [N]
Loop:  CMP     A, D
      BRG     End
Add:   ADD    B, A
      ADDI   A, 1
      JUMP   Loop
End:   STORE  [sum], B
```

Code Memory:

```
0011_01_dd_00000000
0011_00_dd_00000001
1000_11_dd_00000000
1101_00_11_dddddddd
1111_dd_10_00000011
0100_01_00_dddddddd
0101_00_dd_00000001
1110_dd_dd_11110111
1010_01_dd_00000010
```

... are mapped to 0 by the Assembler

.data

```
N      BYTE  5
i      BYTE  ?
sum    BYTE  ?
```

Data Memory:

```
00000101
00000000
00000000
```

.code

```
      LOADI  B, 0
      LOADI  A, 1
      LOAD   D, [N]
Loop:  CMP    A, D
      BRG   End
Add:   ADD   B, A
      ADDI  A, 1
      JUMP  Loop
End:   STORE [sum], B
```

Code Memory:

```
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

Mapping Assembly to Machine Code

.data

N **BYTE** **5**
i **BYTE** **?**
sum **BYTE** **?**

Data Memory:

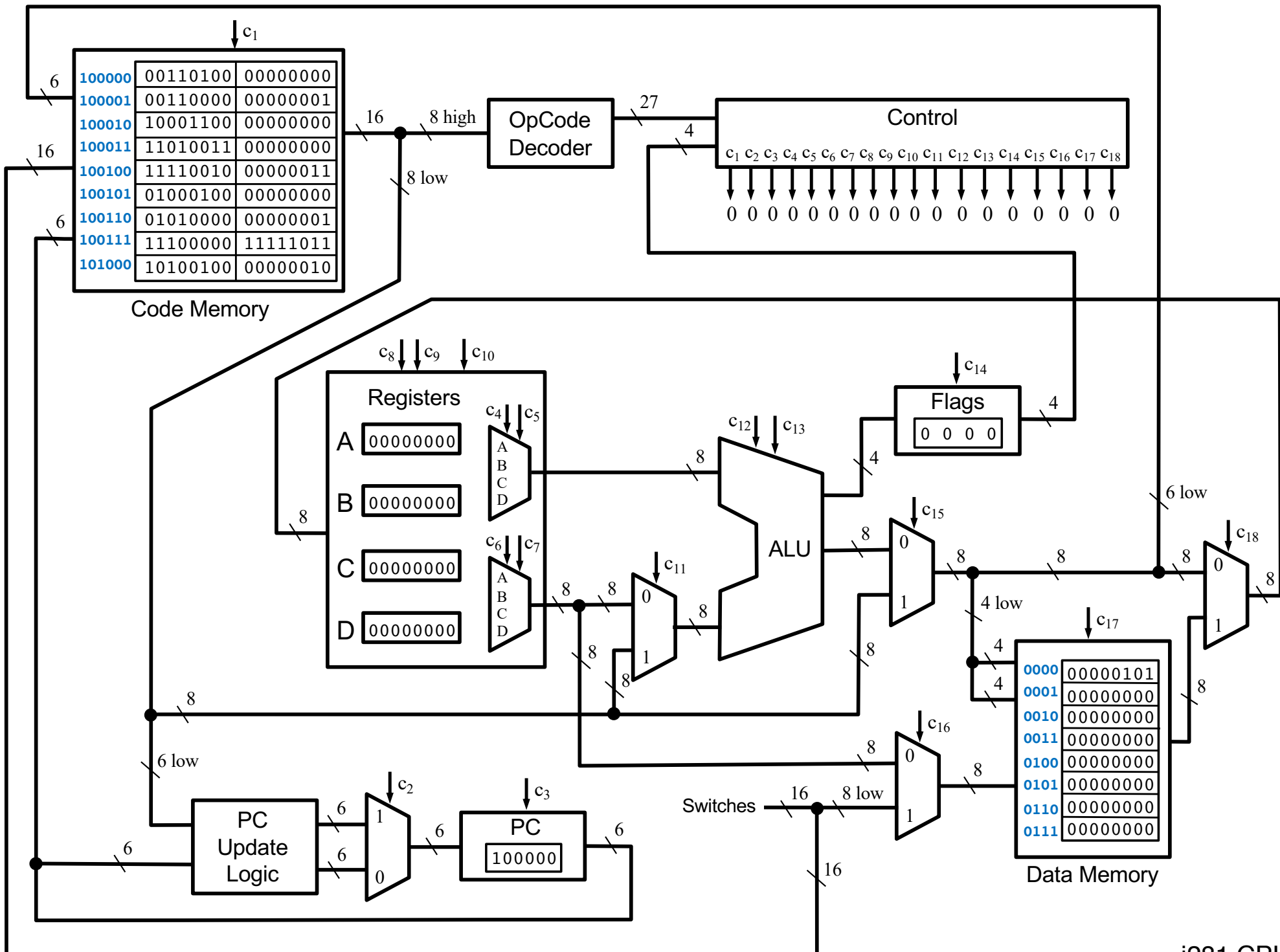
00000101
00000000
00000000

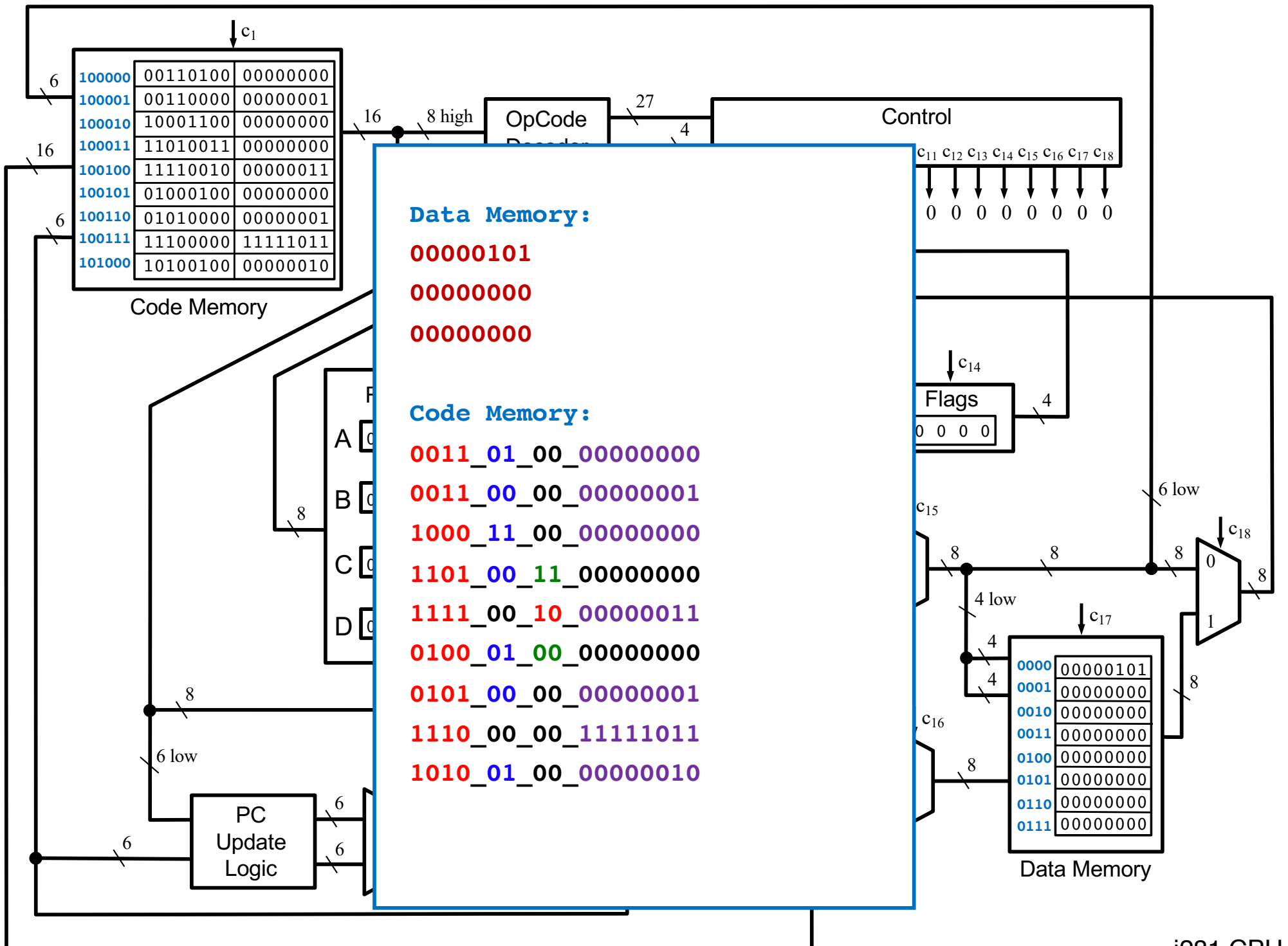
.code

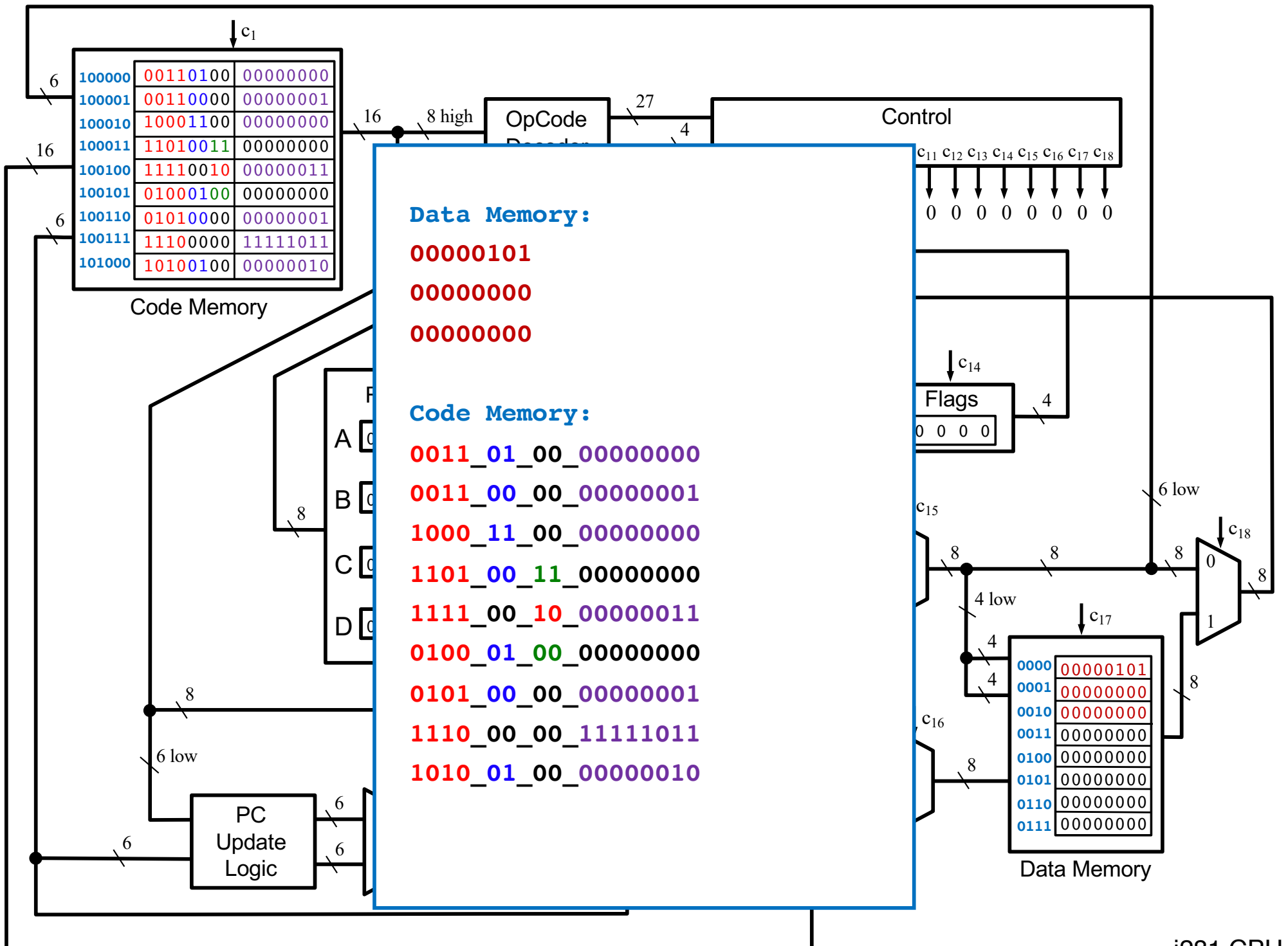
LOADI **B, 0**
 LOADI **A, 1**
 LOAD **D, [N]**
Loop: **CMP** **A, D**
 BRG **End**
Add: **ADD** **B, A**
 ADDI **A, 1**
 JUMP **Loop**
End: **STORE** **[sum], B**

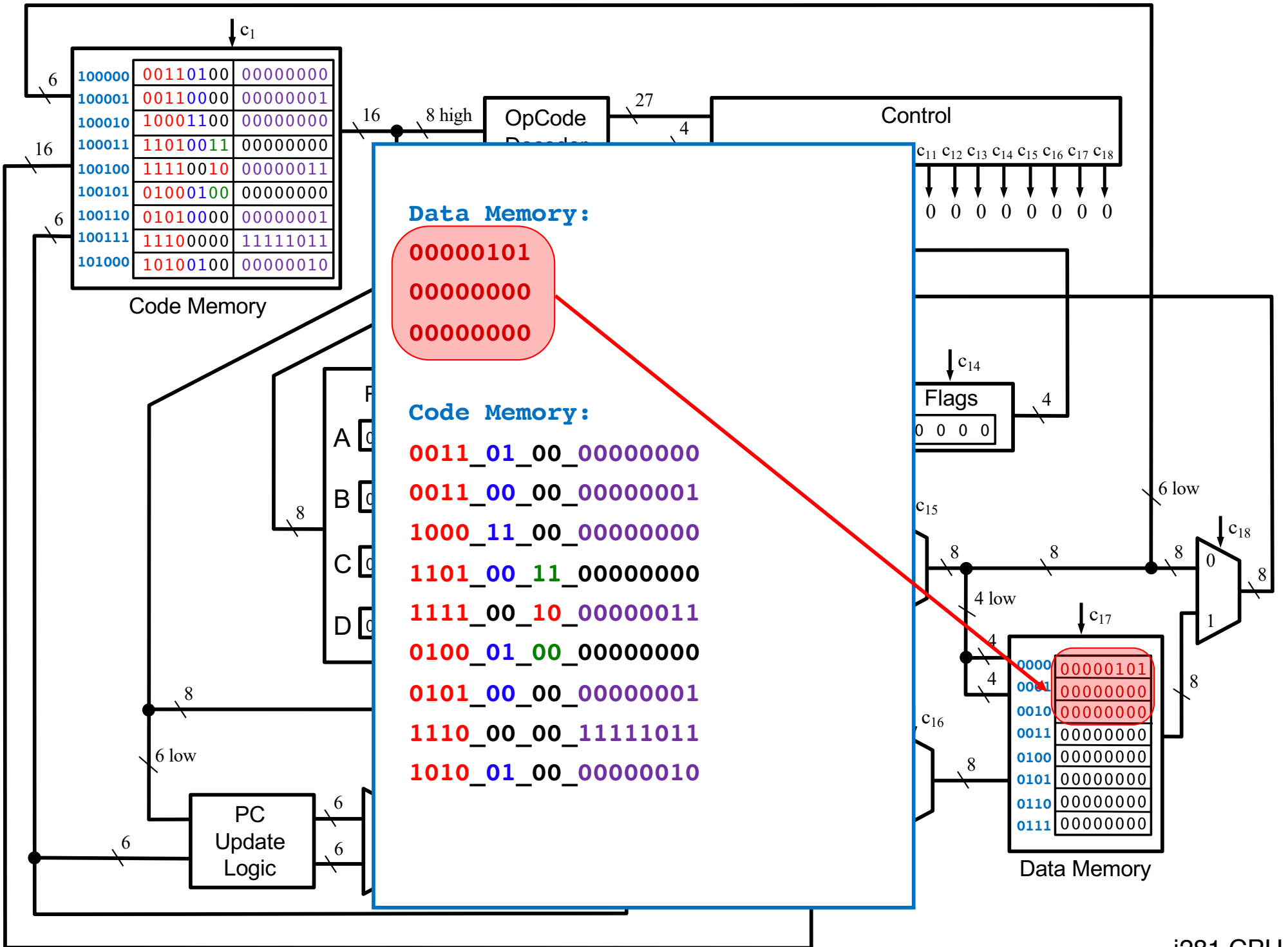
Code Memory:

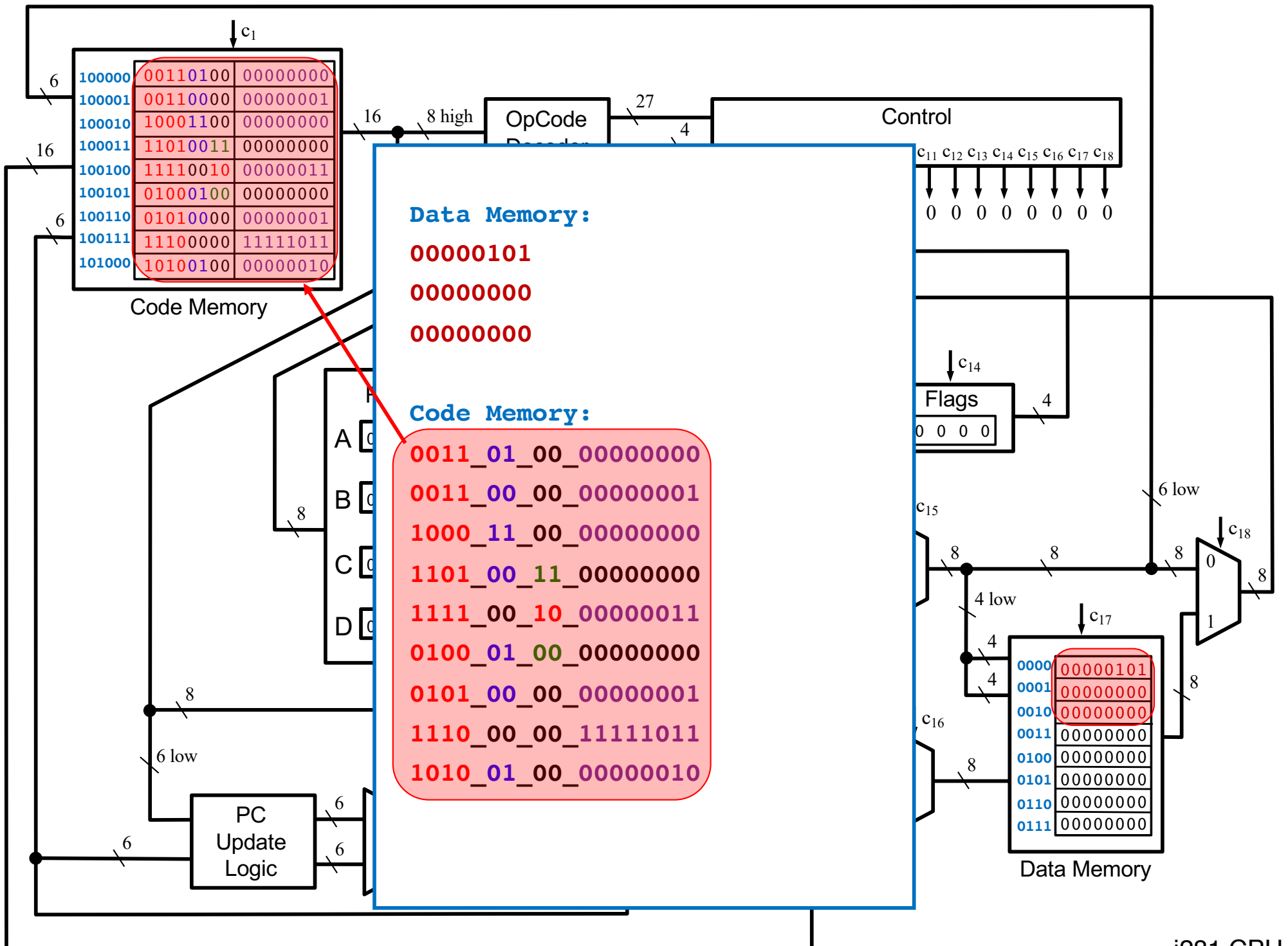
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

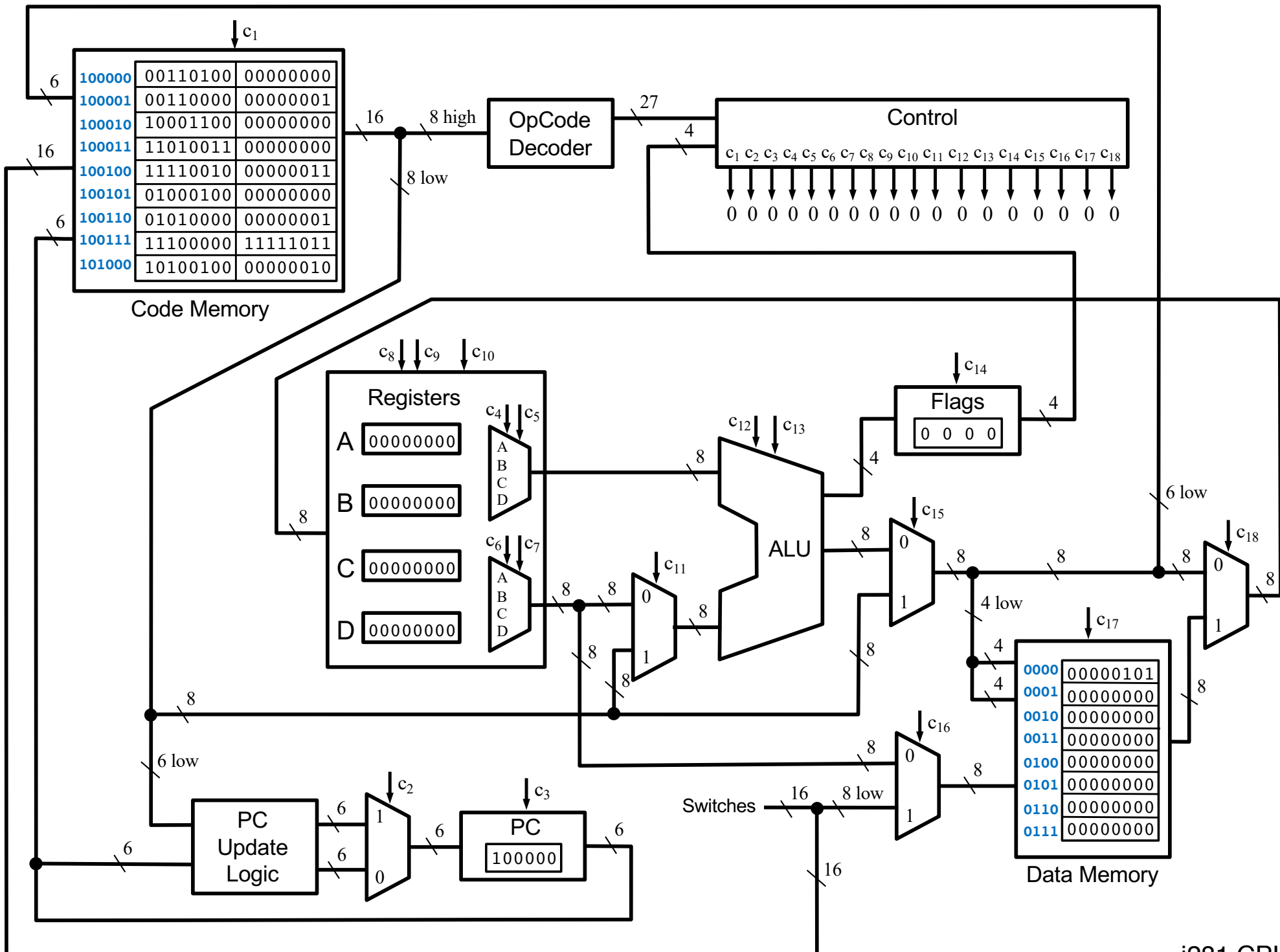












Do Loop Example:
Add the numbers from 1 to 5

C Version

```
// Add the numbers from 1 to 5 using a do loop.
```

```
int N=5;
```

```
int main()
```

```
{
```

```
    int i, sum;
```

```
    i=0;
```

```
    sum=0;
```

```
    do
```

```
    {
```

```
        i++;
```

```
        sum+=i;
```

```
    }while( i < N );
```

```
}
```

Assembly Version

```
; Add the numbers from 1 to 5 using a do loop.
```

```
.data
```

```
N      BYTE    5
```

```
sum    BYTE    ?
```

```
.code
```

```
    LOADI A, 0      ; i = 0
```

```
    LOADI B, 0      ; sum=0
```

```
    LOAD D, [N]     ; register D = N
```

```
Do:  ADDI A, 1      ; i++
```

```
    ADD B, A        ; sum+=i
```

```
    CMP D, A        ; N > i ? (register ordering is swapped)
```

```
    BRG Do          ; if true, jump to Do
```

```
End:  STORE [sum], B ; store sum to memory
```

```
; Register allocation:
```

```
; A: i (the variable i is optimized to register A)
```

```
; B: sum
```

```
; C: <not used>
```

```
; D: N
```

Do Loop

```
// Add the numbers from
// 1 to 5 using a do loop
int N=5;

int main()
{
    int i, sum;

    i=0;
    sum=0;

    do
    {
        i++;
        sum+=i;
    }while( i < N );
}
```

```
; assembly version

.data
N        BYTE    5
sum      BYTE    ?

.code

        LOADI A, 0           ; i = 0
        LOADI B, 0           ; sum=0
        LOAD D, [N]          ; register D = N
Do:     ADDI A, 1             ; i++
        ADD B, A              ; sum+=i
        CMP D, A              ; N > i ?
        BRG Do                ; if true, jump to Do
End:    STORE [sum], B       ; store sum to memory

; Register allocation:
; A: i (the variable i is optimized to register A)
; B: sum
; C: <not used>
; D: N
```

Machine Code Version

Data Memory:

00000101

00000000

Code Memory:

0011000000000000

0011010000000000

1000110000000000

0101000000000001

0100010000000000

1101110000000000

1111001011111100

1010010000000001

Assembly v.s. Machine Code

.data

N BYTE 5

sum BYTE ?

.code

LOADI A, 0

LOADI B, 0

LOAD D, [N]

Do: ADDI A, 1

ADD B, A

CMP D, A

BRG Do

End: STORE [sum], B

Data Memory:

00000101

00000000

Code Memory:

0011000000000000

0011010000000000

1000110000000000

0101000000000001

0100010000000000

1101110000000000

1111001011111100

1010010000000001

Assembly v.s. Machine Code

.data

N **BYTE** **5**
sum **BYTE** **?**

.code

LOADI **A, 0**
 LOADI **B, 0**
 LOAD **D, [N]**
Do: **ADDI** **A, 1**
 ADD **B, A**
 CMP **D, A**
 BRG **Do**
End: **STORE** **[sum], B**

Data Memory:

00000101
00000000

Code Memory:

00110000_00000000
00110100_00000000
10001100_00000000
01010000_00000001
01000100_00000000
11011100_00000000
11110010_11111100
10100100_00000001

Assembly v.s. Machine Code

.data

N **BYTE** **5**
sum **BYTE** **?**

Data Memory:

00000101
00000000

.code

LOADI **A, 0**
 LOADI **B, 0**
 LOAD **D, [N]**
Do: **ADDI** **A, 1**
 ADD **B, A**
 CMP **D, A**
 BRG **Do**
End: **STORE** **[sum], B**

Code Memory:

0011_00_00_00000000
0011_01_00_00000000
1000_11_00_00000000
0101_00_00_00000001
0100_01_00_00000000
1101_11_00_00000000
1111_00_10_11111100
1010_01_00_00000001

Assembly v.s. Machine Code

.data

N BYTE 5

sum BYTE ?

.code

LOADI A, 0

LOADI B, 0

LOAD D, [N]

Do: ADDI A, 1

ADD B, A

CMP D, A

BRG Do

End: STORE [sum], B

Data Memory:

00000101

00000000

Code Memory:

0011_00_00_00000000

0011_01_00_00000000

1000_11_00_00000000

0101_00_00_00000001

0100_01_00_00000000

1101_11_00_00000000

1111_00_10_11111100

1010_01_00_00000001

Assembly v.s. Machine Code

.data

N **BYTE** **5**
sum **BYTE** **?**

Data Memory:

00000101
00000000

.code

LOADI **A, 0**
 LOADI **B, 0**
 LOAD **D, [N]**
Do: **ADDI** **A, 1**
 ADD **B, A**
 CMP **D, A**
 BRG **Do**
End: **STORE** **[sum], B**

Code Memory:

0011_00_00_00000000
0011_01_00_00000000
1000_11_00_00000000
0101_00_00_00000001
0100_01_00_00000000
1101_11_00_00000000
1111_00_10_11111100
1010_01_00_00000001

Assembly v.s. Machine Code

.data

N **BYTE** **5**
sum **BYTE** **?**

Data Memory:

00000101
00000000

.code

LOADI **A, 0**
 LOADI **B, 0**
 LOAD **D, [N]**
Do: **ADDI** **A, 1**
 ADD **B, A**
 CMP **D, A**
 BRG **Do**
End: **STORE** **[sum], B**

Code Memory:

0011_00_00_00000000
0011_01_00_00000000
1000_11_00_00000000
0101_00_00_00000001
0100_01_00_00000000
1101_11_00_00000000
1111_00_10_11111100
1010_01_00_00000001

Assembly v.s. Machine Code

.data

N **BYTE** **5**
sum **BYTE** **?**

Data Memory:

00000101
00000000

.code

LOADI **A**, **0**
 LOADI **B**, **0**
 LOAD **D**, **[N]**
Do: **ADDI** **A**, **1**
 ADD **B**, **A**
 CMP **D**, **A**
 BRG **Do**
End: **STORE** **[sum]**, **B**

Code Memory:

0011_00_00_00000000
0011_01_00_00000000
1000_11_00_00000000
0101_00_00_00000001
0100_01_00_00000000
1101_11_00_00000000
1111_00_10_11111100
1010_01_00_00000001

Assembly v.s. Machine Code

.data

N **BYTE** **5**

sum **BYTE** **?**

.code

LOADI **A**, **0**

LOADI **B**, **0**

LOAD **D**, [**N**]

Do: **ADDI** **A**, **1**

ADD **B**, **A**

CMP **D**, **A**

BRG **Do**

End: **STORE** [**sum**], **B**

Data Memory:

00000101

00000000

Code Memory:

0011_00_00_00000000

0011_01_00_00000000

1000_11_00_00000000

0101_00_00_00000001

0100_01_00_00000000

1101_11_00_00000000

1111_00_10_11111100

1010_01_00_00000001

Assembly v.s. Machine Code

.data

N **BYTE** **5**
sum **BYTE** **?**

Data Memory:

00000101
00000000

.code

LOADI **A, 0**
LOADI **B, 0**
LOAD **D, [N]**
Do: **ADDI** **A, 1**
 ADD **B, A**
 CMP **D, A**
 BRG **Do**
End: **STORE** **[sum], B**

Code Memory:

0011_00_00_00000000
0011_01_00_00000000
1000_11_00_00000000
0101_00_00_00000001
0100_01_00_00000000
1101_11_00_00000000
1111_00_10_11111100
1010_01_00_00000001

Assembly v.s. Machine Code

.data

N **BYTE** **5**

sum **BYTE** **?**

.code

LOADI **A**, **0**

LOADI **B**, **0**

LOAD **D**, [**N**]

Do: **ADDI** **A**, **1**

ADD **B**, **A**

CMP **D**, **A**

BRG **Do**

End: **STORE** [**sum**], **B**

Data Memory:

00000101

00000000

Code Memory:

0011_00_dd_00000000

0011_01_dd_00000000

1000_11_dd_00000000

0101_00_dd_00000001

0100_01_00_00000000

1101_11_00_00000000

1111_dd_10_11111100

1010_01_dd_00000001

Assembly v.s. Machine Code

.data

N **BYTE** **5**
sum **BYTE** **?**

Data Memory:

00000101
00000000

.code

LOADI **A, 0**
LOADI **B, 0**
LOAD **D, [N]**
Do: **ADDI** **A, 1**
 ADD **B, A**
 CMP **D, A**
 BRG **Do**
End: **STORE** **[sum], B**

Code Memory:

0011_00_00_00000000
0011_01_00_00000000
1000_11_00_00000000
0101_00_00_00000001
0100_01_00_00000000
1101_11_00_00000000
1111_00_10_11111100
1010_01_00_00000001

While Loop Example:
Add the numbers from 1 to 5

While Loop

```
// Add the numbers from
// 1 to 5 using a while loop
int N=5;
int sum;
int i;

int main()
{
    i=1;
    sum=0;

    while( i <= N )
    {
        sum+=i;
        i++;
    }
}
```

```
; assembly version
.data
N        BYTE    5
sum      BYTE    ?
; i is optimized to register A

.code
        LOADI  A, 1        ; i = 1
        LOADI  B, 0        ; sum=0
        LOAD   D, [N]      ; register D = N
While:  CMP    A, D        ; i <= N ?
        BRG    End        ; if no, exit the loop
        ADD   B, A        ; sum+=i
        ADDI  A, 1        ; i++
        JUMP  While      ; next iteration
End:    STORE [sum], B    ; store sum to memory

; Register allocation:
; A: i
; B: sum
; C: <not used>
; D: N
```

Bubble Sort

C Version

```
int array[] = {7, 3, 2, 1, 6, 4, 5, 8};
int last = 7; // last valid index in the array
int temp;
int i, j;

int main()
{
    for (i = 0; i < last; i++)
        for (j = 0; j < last-i; j++)
            if (array[j] > array[j+1]){
                temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
            }

    //for(i = 0; i < N; i++){
    //    printf("%d, ", array[i]);
    //}
}
```

Assembly Version

```
.data
array  BYTE 7, 3, 2, 1, 6, 4, 5, 8
last   BYTE 7
temp   BYTE ?

.code

        LOADI  A, 0                ; i = 0;
Outer:  LOAD   D, [last]            ; Load last into D
        LOADI  B, 0                ; j = 0;
        CMP    A, D                ; i < last
        BRGE   End                ; If i >= last break out of the outer loop
Inner:  LOAD   D, [last]            ; Re-Load last into D (this register is shared)
        SUB    D, A                ; D = D - A (i.e., D = last - i)
        CMP    B, D                ; j < last - i
        BRGE   Iinc               ; If j >= last-i branch to Iinc
If:     LOADF  C, [array+B]         ; C = array[j]
        LOADF  D, [array+B+1]      ; D = array[j+1] (compiler adds 1 to addr. of array)
        CMP    D, C                ; if array[j+1] < array[j] (switched direction)
        BRGE   Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1                ; j++
        JUMP   Inner
Iinc:   ADDI   A, 1                ; i++
        JUMP   Outer
End:    NOOP                       ; Do nothing

; Register allocation:
; A: i
; B: j
; C: array[j]
; D: last, array[j+1]

; Notes: i and j are optimized away. They exist only in registers, not in the main memory.
```

Assembly Version

```
.data
array  BYTE 7, 3, 2, 1, 6, 4, 5, 8
last   BYTE 7
temp   BYTE ?

.code

        LOADI  A, 0                ; i = 0;
Outer:  LOAD   D, [last]            ; Load last into D
        LOADI  B, 0                ; j = 0;
        CMP   A, D                 ; i < last
        BRGE  End                  ; If i >= last break out of the outer loop
Inner:  LOAD   D, [last]            ; Re-Load last into D (this register is shared)
        SUB   D, A                 ; D = D - A (i.e., D = last - i)
        CMP   B, D                 ; j < last - i
        BRGE  Iinc                 ; If j >= last-i branch to Iinc
If:     LOADF  C, [array+B]         ; C = array[j]
        LOADF  D, [array+B+1]      ; D = array[j+1] (compiler adds 1 to addr. of array)
        CMP   D, C                 ; if array[j+1] < array[j] (switched direction)
        BRGE  Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI  B, 1                  ; j++
        JUMP  Inner
Iinc:   ADDI  A, 1                  ; i++
        JUMP  Outer
End:    NOOP                        ; Do nothing

; Register allocation:
; A: i
; B: j
; C: array[j]
; D: last, array[j+1]

; Notes: i and j are optimized away. They exist only in registers, not in the main memory.
```

Bubble Sort

```
int array[] = {7, 3, 2, 1, 6, 4, 5, 8};
int last = 7; // last valid index
int temp;
int i, j;

int main()
{
    for (i = 0; i < last; i++)
        for (j = 0; j < last-i; j++)
            if (array[j] > array[j+1]){
                temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
            }
    //for(i = 0; i < N; i++){
    //    printf("%d, ", array[i]);
    //}
}
```

```
.data
array    BYTE 7, 3, 2, 1, 6, 4, 5, 8
last     BYTE 7
temp     BYTE ?

.code

        LOADI   A, 0                ; i = 0;
Outer:  LOAD    D, [last]           ; Load last into D
        LOADI   B, 0                ; j = 0;
        CMP     A, D                ; i < last
        BRGE   End                 ; If i >= last
Inner:  LOAD    D, [last]           ; Re-Load last into D
        SUB    D, A                ; D=D-A, i.e., D=last-i
        CMP    B, D                ; j < last - i
        BRGE   Iinc                ; If j >= last-i
If:     LOADF   C, [array+B]        ; C = array[j]
        LOADF   D, [array+B+1]     ; D = array[j+1]
        CMP    D, C                ; array[j+1] < array[j]
        BRGE   Jinc                ;
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1                ; j++
        JUMP   Inner
Iinc:   ADDI   A, 1                ; i++
        JUMP   Outer
End:    NOOP

; Register allocation:
; A: i
; B: j
; C: array[j]
; D: last, array[j+1] (shared)

; Notes: i and j are optimized away to registers.
```


Machine Code Version

```
.data
array  BYTE 7, 3, 2, 1, 6, 4, 5, 8
last   BYTE 7
temp   BYTE ?
```

```
.code

        LOADI  A, 0
Outer:  LOAD   D, [last]
        LOADI  B, 0
        CMP    A, D
        BRGE   End
Inner:  LOAD   D, [last]
        SUB    D, A
        CMP    B, D
        BRGE   Iinc
If:     LOADF  C, [array+B]
        LOADF  D, [array+B+1]
        CMP    D, C
        BRGE   Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1
        JUMP   Inner
Iinc:   ADDI   A, 1
        JUMP   Outer
End:    NOOP
```

Machine Code Version

		Data Memory:
.data		
array	BYTE 7, 3, 2, 1, 6, 4, 5, 8	00000111
last	BYTE 7	00000011
temp	BYTE ?	00000010
.code		00000001
	LOADI A, 0	00000110
Outer:	LOAD D, [last]	00000100
	LOADI B, 0	00000101
	CMP A, D	00001000
	BRGE End	00000111
Inner:	LOAD D, [last]	00000000
	SUB D, A	
	CMP B, D	
	BRGE Iinc	
If:	LOADF C, [array+B]	
	LOADF D, [array+B+1]	
	CMP D, C	
	BRGE Jinc	
Swap:	STOREF [array+B], D	
	STOREF [array+B+1], C	
Jinc:	ADDI B, 1	
	JUMP Inner	
Iinc:	ADDI A, 1	
	JUMP Outer	
End:	NOOP	

Machine Code Version

```
.data
array    BYTE 7, 3, 2, 1, 6, 4, 5, 8
last     BYTE 7
temp     BYTE ?

.code

        LOADI  A, 0
Outer:  LOAD    D, [last]
        LOADI  B, 0
        CMP    A, D
        BRGE  End
Inner:  LOAD    D, [last]
        SUB    D, A
        CMP    B, D
        BRGE  Iinc
If:     LOADF  C, [array+B]
        LOADF  D, [array+B+1]
        CMP    D, C
        BRGE  Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1
        JUMP  Inner
Iinc:   ADDI   A, 1
        JUMP  Outer
End:    NOOP
```

Data Memory:

```
00000111 //array[0]
00000011 //array[1]
00000010 //array[2]
00000001 //array[3]
00000110 //array[4]
00000100 //array[5]
00000101 //array[6]
00001000 //array[7]
00000111 //last
00000000 //temp
```

Machine Code Version

		Address	Data Memory:
.data			
array	BYTE 7, 3, 2, 1, 6, 4, 5, 8	0000	00000111 //array[0]
last	BYTE 7	0001	00000011 //array[1]
temp	BYTE ?	0010	00000010 //array[2]
		0011	00000001 //array[3]
.code		0100	00000110 //array[4]
Outer:	LOADI A, 0	0101	00000100 //array[5]
	LOAD D, [last]	0110	00000101 //array[6]
	LOADI B, 0	0111	00001000 //array[7]
	CMP A, D	1000	00000111 //last
	BRGE End	1001	00000000 //temp
Inner:	LOAD D, [last]		
	SUB D, A		
	CMP B, D		
	BRGE Iinc		
If:	LOADF C, [array+B]		
	LOADF D, [array+B+1]		
	CMP D, C		
	BRGE Jinc		
Swap:	STOREF [array+B], D		
	STOREF [array+B+1], C		
Jinc:	ADDI B, 1		
	JUMP Inner		
Iinc:	ADDI A, 1		
	JUMP Outer		
End:	NOOP		

Machine Code Version

		Address	Data Memory:
.data			
array	BYTE 7, 3, 2, 1, 6, 4, 5, 8	0000	00000111 //array[0]
last	BYTE 7	0001	00000011 //array[1]
temp	BYTE ?	0010	00000010 //array[2]
		0011	00000001 //array[3]
.code		0100	00000110 //array[4]
Outer:	LOADI A, 0	0101	00000100 //array[5]
	LOAD D, [last]	0110	00000101 //array[6]
	LOADI B, 0	0111	00001000 //array[7]
	CMP A, D	1000	00000111 //last
	BRGE End	1001	00000000 //temp
Inner:	LOAD D, [last]	1010	00000000
	SUB D, A	1011	00000000
	CMP B, D	1100	00000000
	BRGE Iinc	1101	00000000
If:	LOADF C, [array+B]	1110	00000000
	LOADF D, [array+B+1]	1111	00000000
	CMP D, C		
	BRGE Jinc		
Swap:	STOREF [array+B], D		
	STOREF [array+B+1], C		
Jinc:	ADDI B, 1		
	JUMP Inner		
Iinc:	ADDI A, 1		
	JUMP Outer		
End:	NOOP		

Machine Code Version

```
.data
array  BYTE 7, 3, 2, 1, 6, 4, 5, 8
last   BYTE 7
temp   BYTE ?
```

```
.code

        LOADI  A, 0
Outer:  LOAD   D, [last]
        LOADI  B, 0
        CMP    A, D
        BRGE   End
Inner:  LOAD   D, [last]
        SUB    D, A
        CMP    B, D
        BRGE   Iinc
If:     LOADF  C, [array+B]
        LOADF  D, [array+B+1]
        CMP    D, C
        BRGE   Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1
        JUMP   Inner
Iinc:   ADDI   A, 1
        JUMP   Outer
End:    NOOP
```

Machine Code Version

```
.data
array  BYTE 7, 3, 2, 1, 6, 4, 5, 8
last   BYTE 7
temp   BYTE ?
```

```
.code
        LOADI  A, 0
Outer:  LOAD   D, [last]
        LOADI  B, 0
        CMP    A, D
        BRGE  End
Inner:  LOAD   D, [last]
        SUB    D, A
        CMP    B, D
        BRGE  Iinc
If:     LOADF  C, [array+B]
        LOADF  D, [array+B+1]
        CMP    D, C
        BRGE  Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1
        JUMP  Inner
Iinc:   ADDI   A, 1
        JUMP  Outer
End:    NOOP
```

Code Memory:

```
0011000000000000
1000110000001000
0011010000000000
1101001100000000
1111001100001110
1000110000001000
0110110000000000
1101011100000000
1111001100001000
1001100100000000
1001110100000001
1101111000000000
1111001100000010
1011110100000000
1011100100000001
0101010000000001
1110000011110100
0101000000000001
1110000011101110
0000000000000000
```


Assembly v.s. Machine Code

		Code Memory:
.code		0011000000000000
	LOADI A, 0	0011000000000000
Outer:	LOAD D, [last]	1000110000001000
	LOADI B, 0	0011010000000000
	CMP A, D	1101001100000000
	BRGE End	1111001100001110
Inner:	LOAD D, [last]	1000110000001000
	SUB D, A	0110110000000000
	CMP B, D	1101011100000000
	BRGE Iinc	1111001100001000
If:	LOADF C, [array+B]	1001100100000000
	LOADF D, [array+B+1]	1001110100000001
	CMP D, C	1101111000000000
	BRGE Jinc	1111001100000010
Swap:	STOREF [array+B], D	1011110100000000
	STOREF [array+B+1], C	1011100100000001
Jinc:	ADDI B, 1	0101010000000001
	JUMP Inner	1110000011110100
Iinc:	ADDI A, 1	0101000000000001
	JUMP Outer	1110000011101110
End:	NOOP	0000000000000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	00110000_00000000
Outer:	LOAD D, [last]	10001100_00001000
	LOADI B, 0	00110100_00000000
	CMP A, D	11010011_00000000
	BRGE End	11110011_00001110
Inner:	LOAD D, [last]	10001100_00001000
	SUB D, A	01101100_00000000
	CMP B, D	11010111_00000000
	BRGE Iinc	11110011_00001000
If:	LOADF C, [array+B]	10011001_00000000
	LOADF D, [array+B+1]	10011101_00000001
	CMP D, C	11011110_00000000
	BRGE Jinc	11110011_00000010
Swap:	STOREF [array+B], D	10111101_00000000
	STOREF [array+B+1], C	10111001_00000001
Jinc:	ADDI B, 1	01010100_00000001
	JUMP Inner	11100000_11110100
Iinc:	ADDI A, 1	01010000_00000001
	JUMP Outer	11100000_11101110
End:	NOOP	00000000_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		0011_00_00_00000000
	LOADI A, 0	
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		0011_00_dd_00000000
	LOADI A, 0	
Outer:	LOAD D, [last]	1000_11_dd_00001000
	LOADI B, 0	0011_01_dd_00000000
	CMP A, D	1101_00_11_dddddddd
	BRGE End	1111_dd_11_00001110
Inner:	LOAD D, [last]	1000_11_dd_00001000
	SUB D, A	0110_11_00_dddddddd
	CMP B, D	1101_01_11_dddddddd
	BRGE Iinc	1111_dd_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_dddddddd
	BRGE Jinc	1111_dd_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_dd_00000001
	JUMP Inner	1110_dd_dd_11110100
Iinc:	ADDI A, 1	0101_00_dd_00000001
	JUMP Outer	1110_dd_dd_11101110
End:	NOOP	0000_dd_dd_dddddddd

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

**For more examples
try the i281 simulator**

i281 Simulator

Current Instruction: **LOADI A, 0** i281 CPU Running: **BubbleSort** [About](#)

Speed: 50

- Auto Mode on
- Game Mode on
- Register View
- Start PC @ 32
- Stop At End
- Show Description
- Show Bus Width
- Syntax Highlighting
- Show Data Path
- Show Control Path

RUN **STEP** **RESET** **LOAD**

To try the simulator, go to the class web page and follow the link.

Questions?

THE END