

1281 Cpu
~~1286~~ -cpu

from Stoytchev's &
Digital Logic course
Cpre 2810

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
```

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);
}
```

Mapping Assembly to Machine Code

.data

N BYTE 5
i BYTE ?
sum BYTE ?

Data Memory:

00000101
00000000
00000000

.code

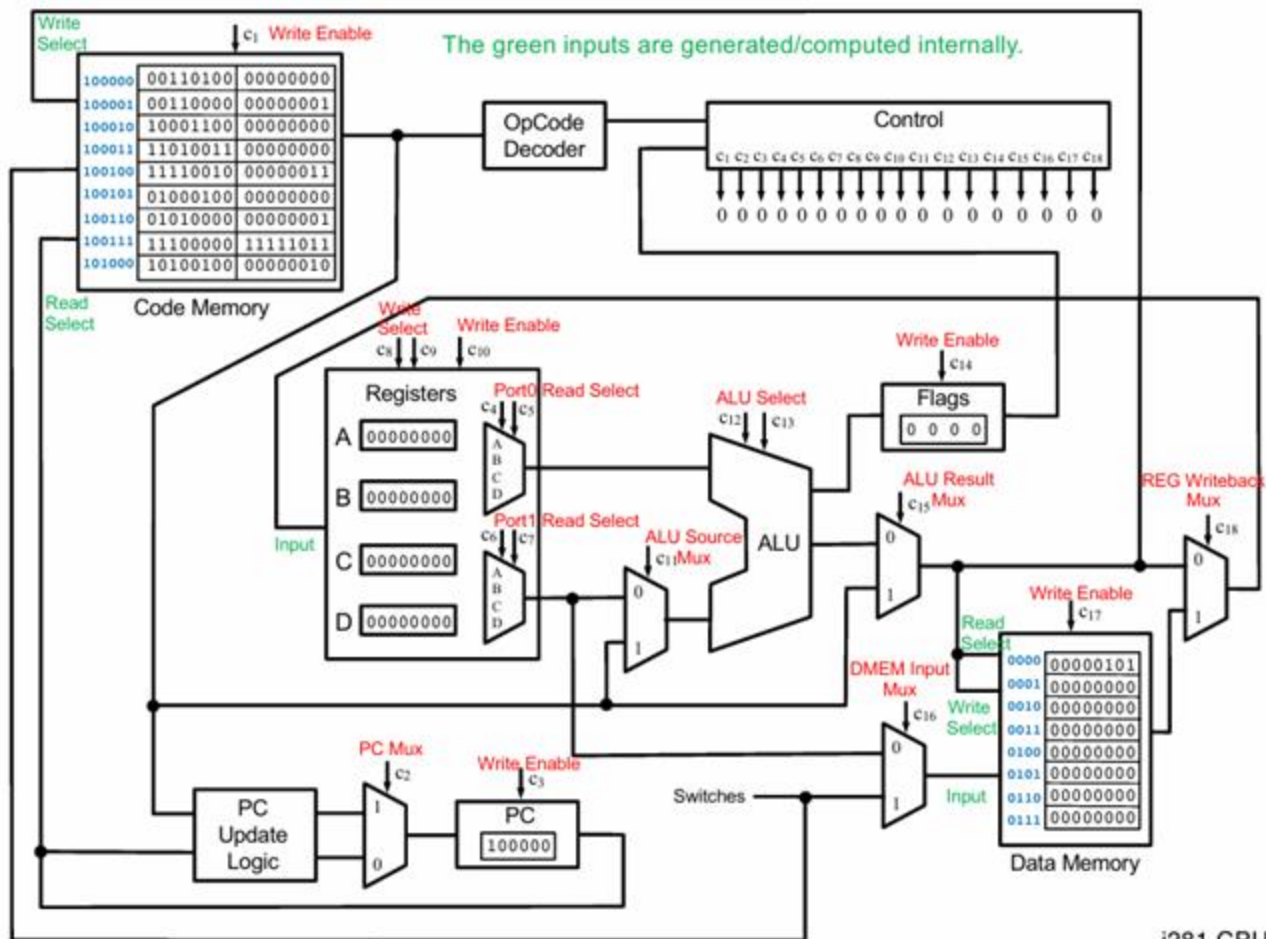
 LOADI B, 0
 LOADI A, 1
 LOAD D, [N]
Loop: CMP A, D
 BRG End
Add: ADD B, A
 ADDI A, 1
 JUMP Loop
End: STORE [sum], B

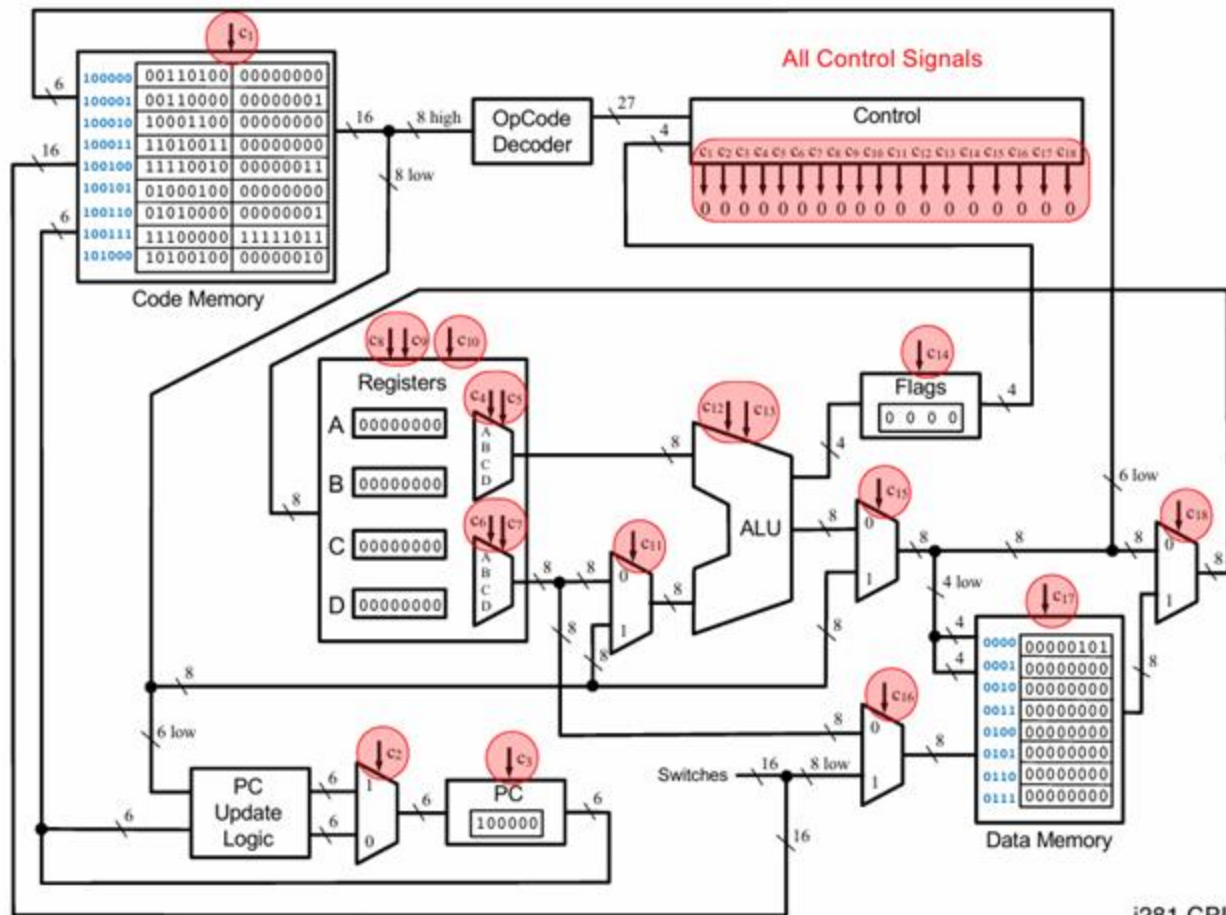
Code Memory:

0011010000000000
0011000000000001
1000110000000000
1101001100000000
1111001000000011
0100010000000000
0101000000000001
1110000011111011
1010010000000010

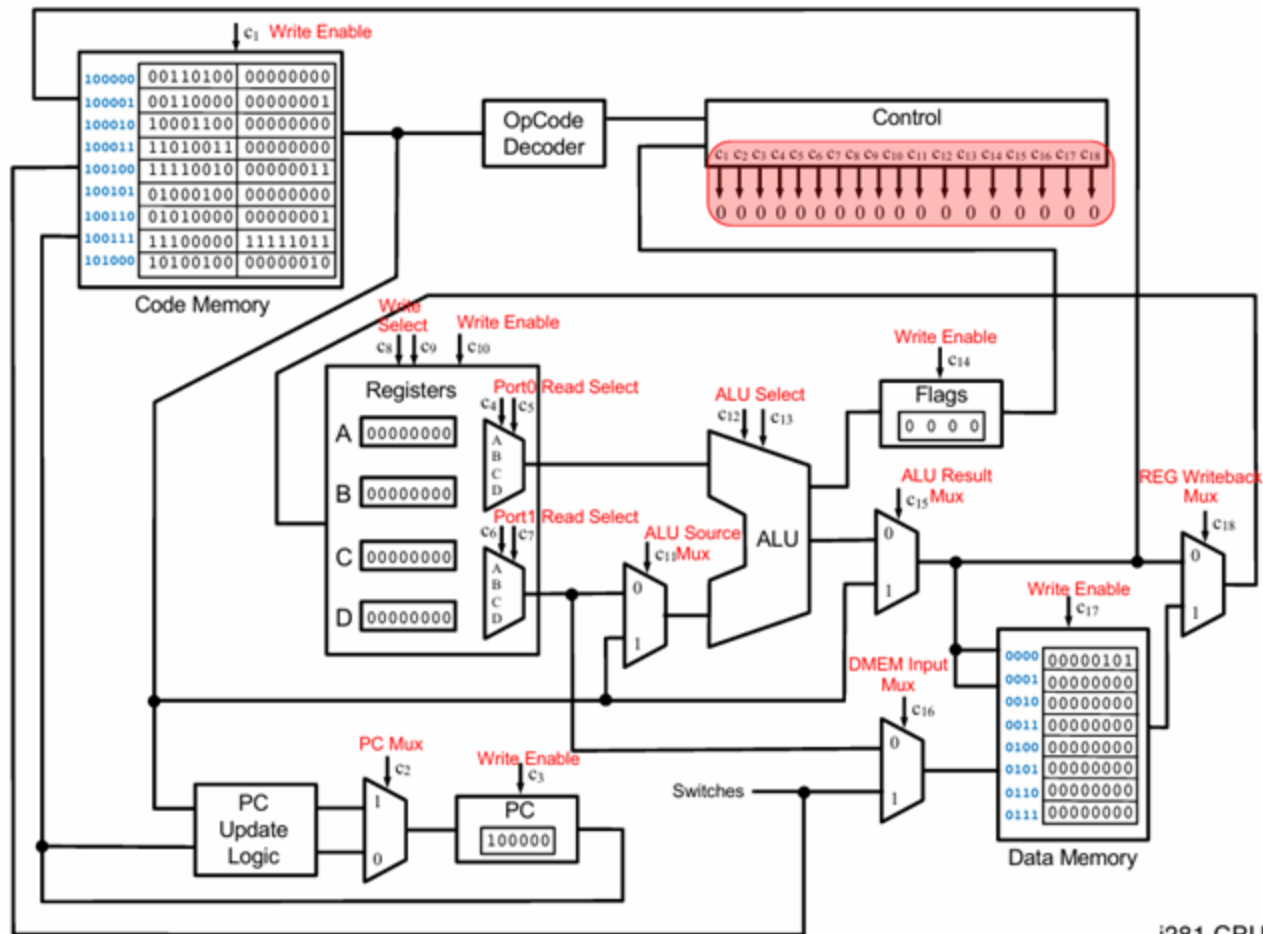
Assembly Language

Machine Language





i281 CPU



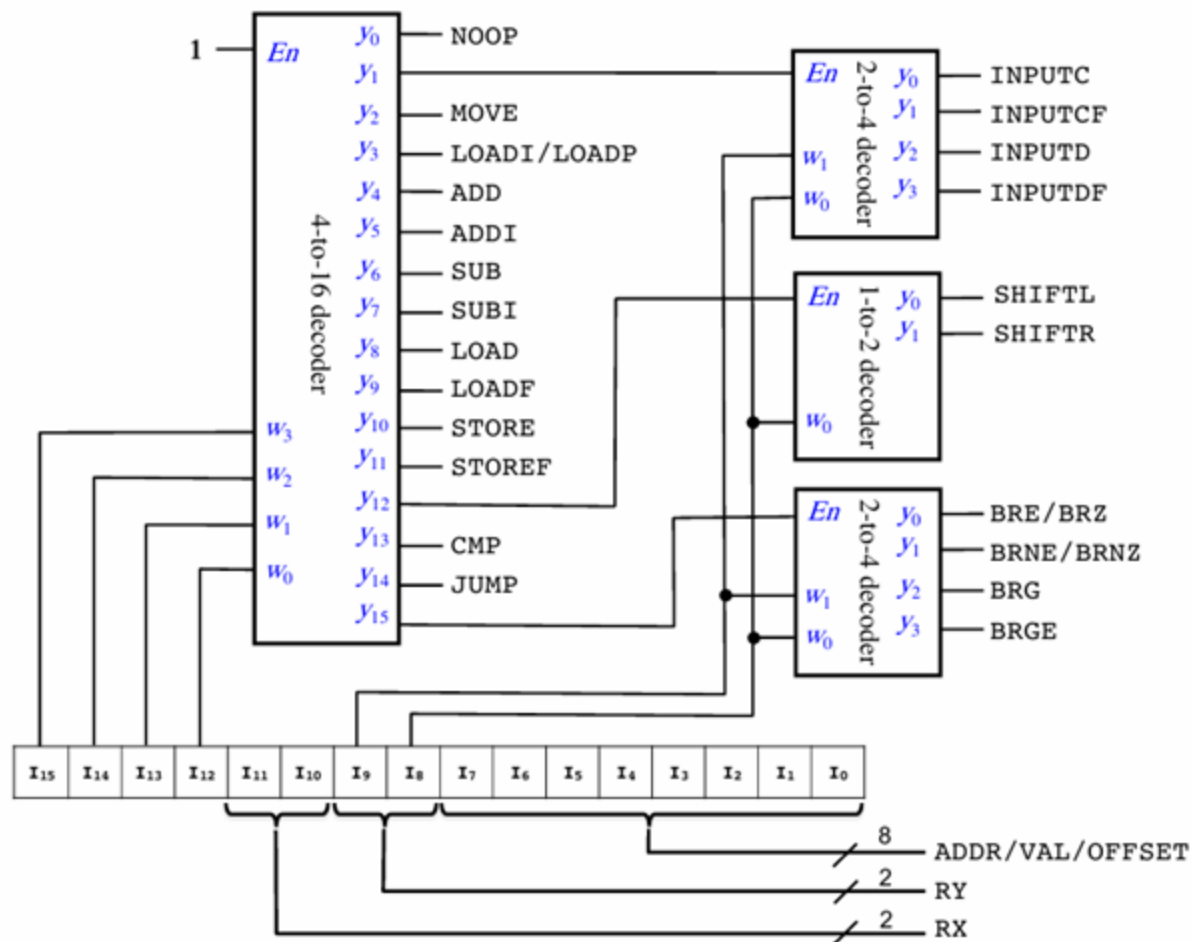
i281 CPU

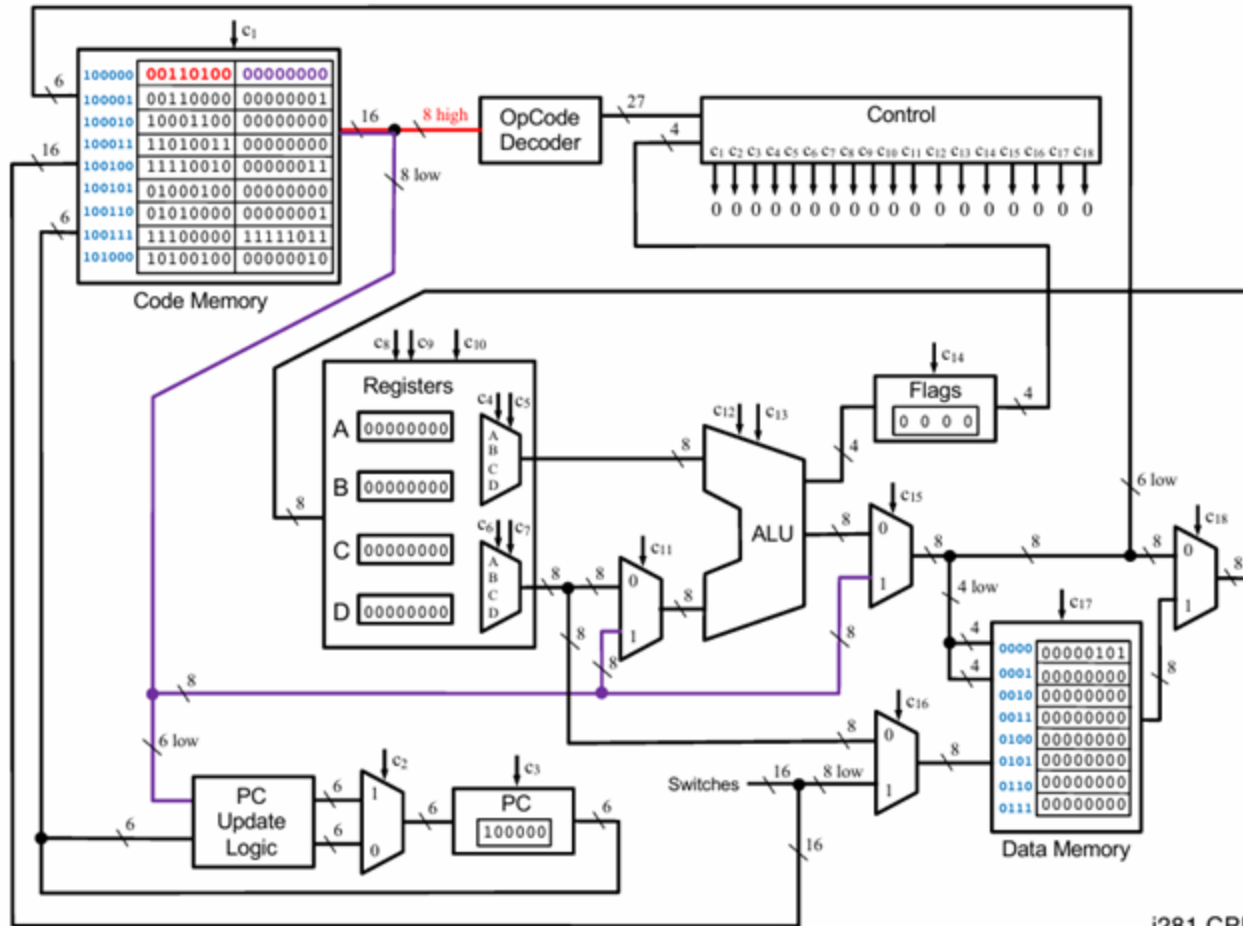
Memory Layout

- The i281 CPU uses two different memories
- Data Memory
16 x 8 bits
- Code Memory
64 x 16 bits

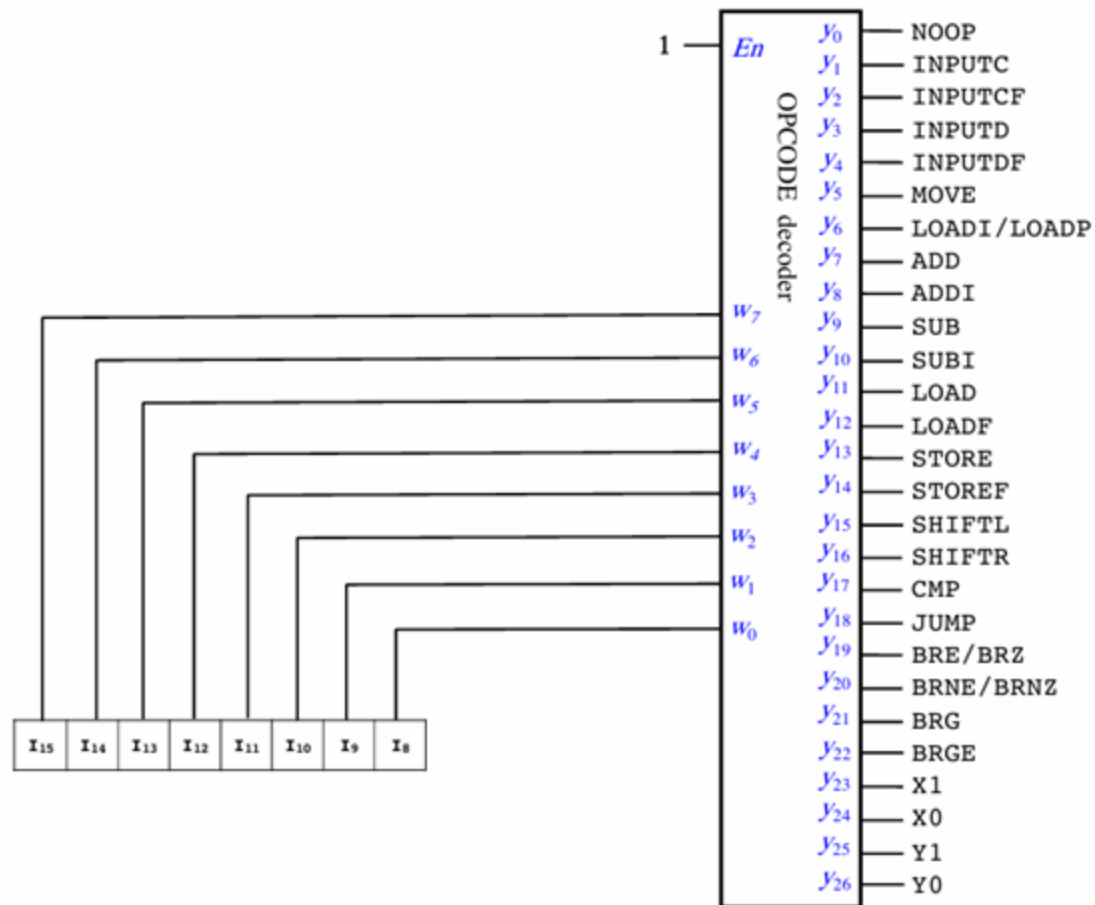
The i281 Assembly Instructions

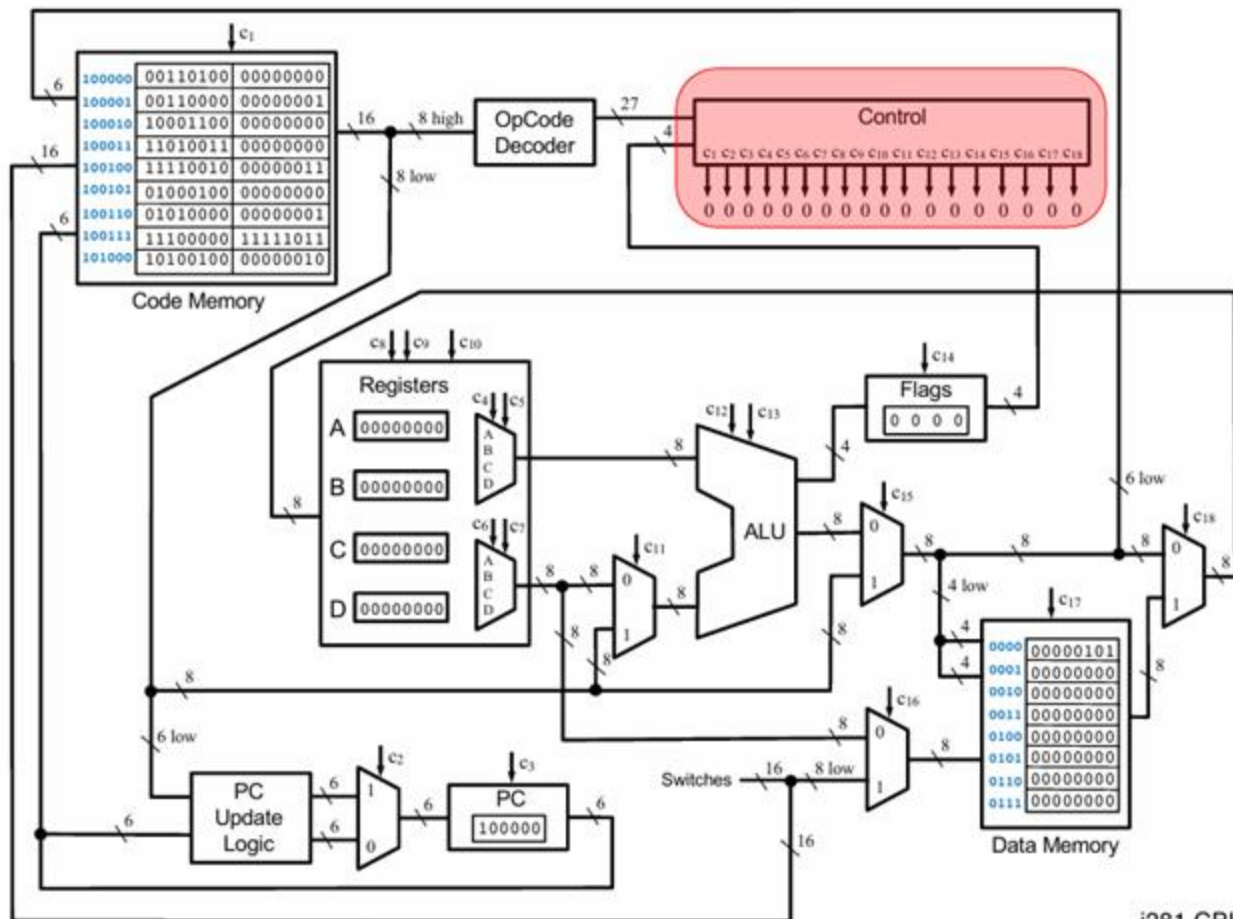
NOOP	NO OPERATION
INPUTC	INPUT into Code memory
INPUTCF	INPUT into Code memory with offset
INPUTD	INPUT into Data memory
INPUTDF	INPUT into Data memory with offset
MOVE	MOVE the contents of one register into another
LOADI	LOAD Immediate value
LOADP	LOAD Pointer address
ADD	ADD two registers
ADDI	ADD 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





i281 CPU





i281 CPU

NOOP
INPUTC
INPUTCF
INPUTD
INPUTDF
MOVE
LOADI/LOADP
ADD
ADDI
SUB
SUBI
LOAD
LOADF
STORE
STOREF
SHIFTL
SHIFTR
CMP
JUMP
BRE/BRZ
BRNE/BRNZ
BRG
BRGE

NOOP
INPUTC
INPUTCF
INPUTD
INPUTDF
MOVE
LOADI/LOADP
ADD
ADDI
SUB
SUBI
LOAD
LOADF
STORE
STOREF
SHIFTL
SHIFTR
CMP
JUMP
BRE/BRZ
BRNE/BRNZ
BRG
BRGE

	C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇	C ₈	C ₉	C ₁₀	C ₁₁	C ₁₂	C ₁₃	C ₁₄	C ₁₅	C ₁₆	C ₁₇	C ₁₈
	INEM_WRITE_ENABLE	PROGRAM_COUNTER_MUX	PROGRAM_COUNTER_WRITE_EN	REGISTERS_PORTO_SELECT1	REGISTERS_PORTO_SELECT0	REGISTERS_PORTI_SELECT1	REGISTERS_PORTI_SELECT0	REGISTERS_WRITE_SELECT1	REGISTERS_WRITE_SELECT0	REGISTERS_WRITE_ENABLE	ALU_SOURCE_MUX	ALU_SELECT1	ALU_SELECT0	FLAGS_WRITE_ENABLE	ALU_RESUT_MUX	DMEM_INFUT_MUX	DMEM_WRITE_ENABLE	REG_WRITEBACK_MUX
B1	1		1												1			
B2			1	X1	X0						1	1						
B3			1	X1	X0										1	1	1	
B4			1	Y1	Y0			X1	X0	1	1	1				1	1	
			1	X1	X0	Y1	Y0	X1	X0	1				1				
			1	X1	X0			X1	X0	1	1	1		1				
			1	X1	X0			X1	X0	1	1	1	1	1				
			1	X1	X0	Y1	Y0	X1	X0	1		1	1	1				
			1	X1	X0			X1	X0	1	1	1	1	1				1
			1	Y1	Y0			X1	X0	1	1	1			1			1
			1			X1	X0									1	1	
			1	Y1	Y0	X1	X0				1	1					1	
			1	X1	X0			X1	X0	1				1				
			1	X1	X0			X1	X0	1			1	1				
			1	X1	X0	Y1	Y0					1	1	1				

[illegible]

Taken from
these bits of the
instruction

[illegible]

	C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇	C ₈	C ₉	C ₁₀	C ₁₁	C ₁₂	C ₁₃	C ₁₄	C ₁₅	C ₁₆	C ₁₇	C ₁₈
NOOP			1															
INPUTC	1		1												1			
INPUTCF	1		1	X1	X0						1	1						
INPUTD			1												1	1	1	
INPUTDF			1	X1	X0						1	1				1	1	
MOVE			1	Y1	Y0			X1	X0	1	1	1						
LOADI/LOADP			1					X1	X0	1					1			
ADD			1	X1	X0	Y1	Y0	X1	X0	1		1		1				
ADDI			1	X1	X0			X1	X0	1	1	1		1				
SUB			1	X1	X0	Y1	Y0	X1	X0	1		1	1	1				
SUBI			1	X1	X0			X1	X0	1	1	1	1	1				
LOAD			1					X1	X0	1					1			1
LOADF			1	Y1	Y0			X1	X0	1	1	1						1
STORE			1			X1	X0								1		1	
STOREF			1	Y1	Y0	X1	X0				1	1					1	
SHIFTL			1	X1	X0			X1	X0	1				1				
SHIFTR			1	X1	X0			X1	X0	1			1	1				
CMP			1	X1	X0	Y1	Y0					1	1	1				
JUMP		1	1															
BRE/BRZ		B1	1															
BRNE/BRNZ		B2	1															
BRG		B3	1															
BRGE		B4	1															

Taken from
these bits of the
instruction

I ₁₅	I ₁₄	I ₁₃	I ₁₂	I ₁₁	I ₁₀	I ₉	I ₈	I ₇	I ₆	I ₅	I ₄	I ₃	I ₂	I ₁	I ₀
						Y ₁	Y ₀								

	C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇	C ₈	C ₉	C ₁₀	C ₁₁	C ₁₂	C ₁₃	C ₁₄	C ₁₅	C ₁₆	C ₁₇	C ₁₈
NOOP			1															
INPUTC	1		1												1			
INPUTCF	1		1	X1	X0						1	1						
INPUTD			1												1	1	1	
INPUTDF			1	X1	X0						1	1				1	1	
MOVE			1	Y1	Y0			X1	X0	1	1	1						
LOADI/LOADP			1					X1	X0	1					1			
ADD			1	X1	X0	Y1	Y0	X1	X0	1		1		1				
ADDI			1	X1	X0			X1	X0	1	1	1		1				
SUB			1	X1	X0	Y1	Y0	X1	X0	1		1	1	1				
SUBI			1	X1	X0			X1	X0	1	1	1	1	1				
LOAD			1					X1	X0	1					1			1
LOADF			1	Y1	Y0			X1	X0	1	1	1						1
STORE			1			X1	X0								1		1	
STOREF			1	Y1	Y0	X1	X0				1	1					1	
SHIFTL			1	X1	X0			X1	X0	1				1				
SHIFTR			1	X1	X0			X1	X0	1			1	1				
CMP			1	X1	X0	Y1	Y0					1	1	1				
JUMP		1	1															
BRE/BRZ		B1	1															
BRNE/BRNZ		B2	1															
BRG		B3	1															
BRGE		B4	1															

Taken from
these bits of the
instruction

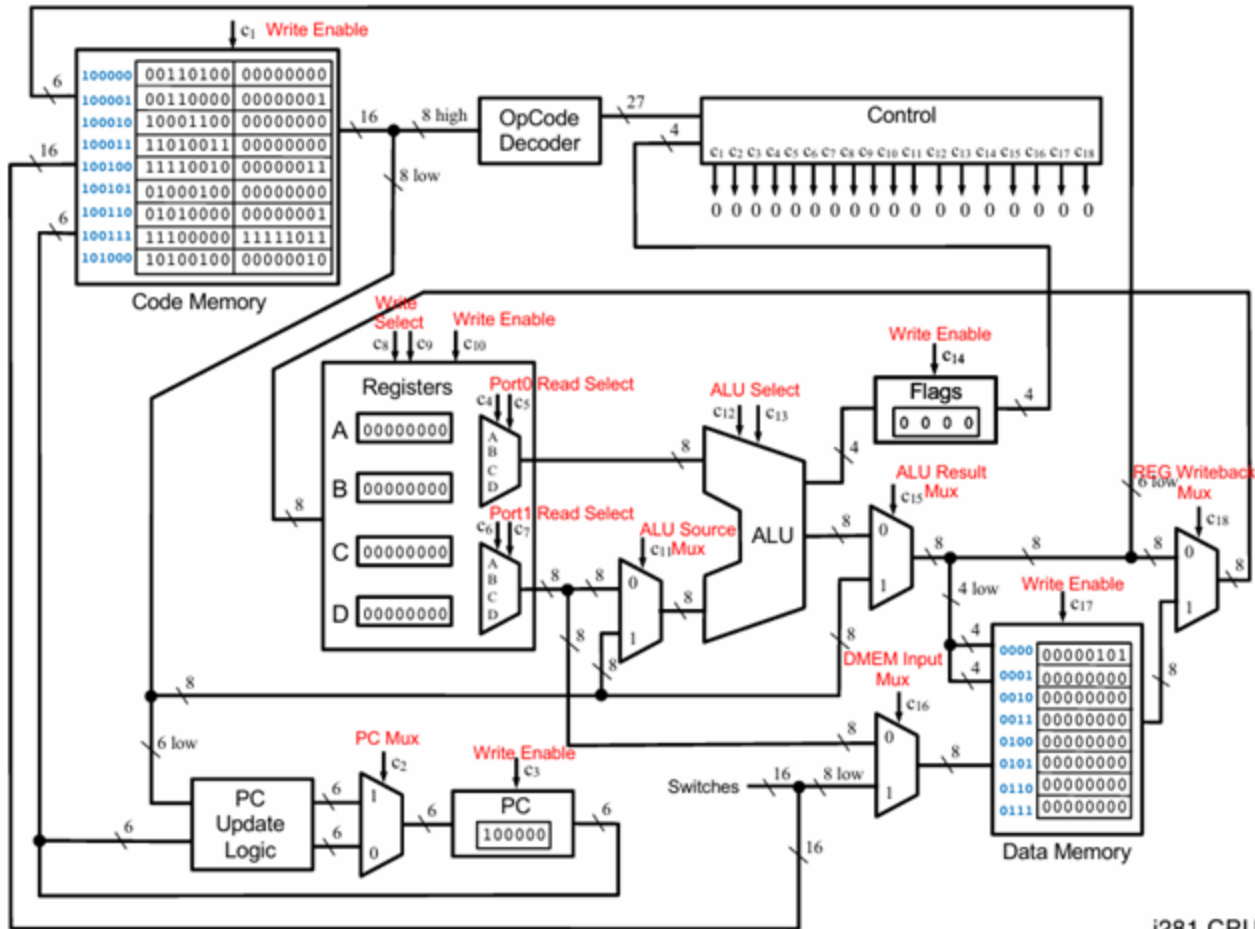
I ₁₅	I ₁₄	I ₁₃	I ₁₂	I ₁₁	I ₁₀	I ₉	I ₈	I ₇	I ₆	I ₅	I ₄	I ₃	I ₂	I ₁	I ₀
						Y ₁	Y ₀								

	C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇	C ₈	C ₉	C ₁₀	C ₁₁	C ₁₂	C ₁₃	C ₁₄	C ₁₅	C ₁₆	C ₁₇	C ₁₈
NOOP			1															
INPUTC	1		1												1			
INPUTCF	1		1	X1	X0						1	1						
INPUTD			1												1	1	1	
INPUTDF			1	X1	X0						1	1				1	1	
MOVE			1	Y1	Y0			X1	X0	1	1	1						
LOADI/LOADP			1					X1	X0	1					1			
ADD			1	X1	X0	Y1	Y0	X1	X0	1		1		1				
ADDI			1	X1	X0			X1	X0	1	1	1		1				
SUB			1	X1	X0	Y1	Y0	X1	X0	1		1	1	1	1			
SUBI			1	X1	X0			X1	X0	1	1	1	1	1				
LOAD			1					X1	X0	1					1			1
LOADF			1	Y1	Y0			X1	X0	1	1	1						1
STORE			1			X1	X0								1		1	
STOREF			1	Y1	Y0	X1	X0				1	1					1	
SHIFTL			1	X1	X0			X1	X0	1				1				
SHIFTR			1	X1	X0			X1	X0	1				1	1			
CMP			1	X1	X0	Y1	Y0							1	1	1		
JUMP		1	1															
BRE/BRZ		B1	1															
BRNE/BRNZ		B2	1															
BRG		B3	1															
BRGE		B4	1															

computed using
the flags register

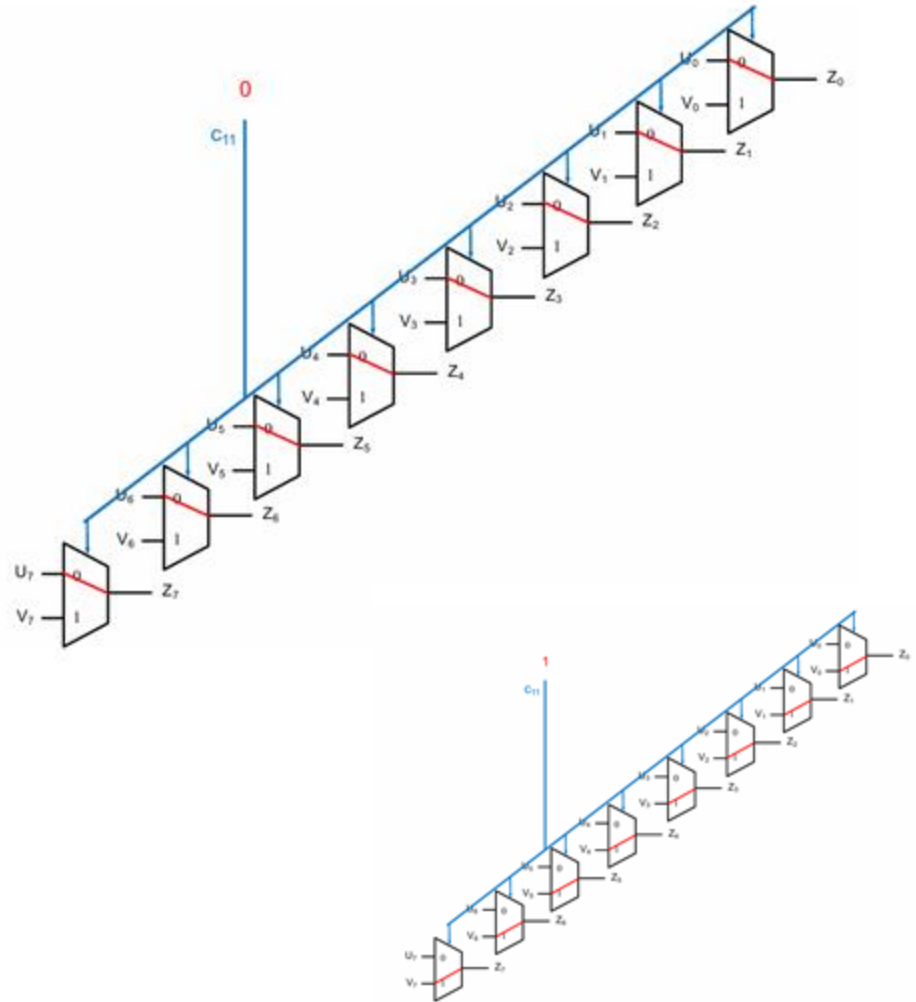
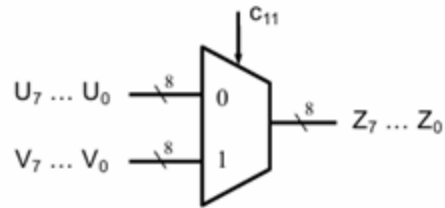
B1= ZF
B2= ~ZF
B3= AND(~ZF, XNOR(NF, OF))
B4= XNOR(NF, OF)

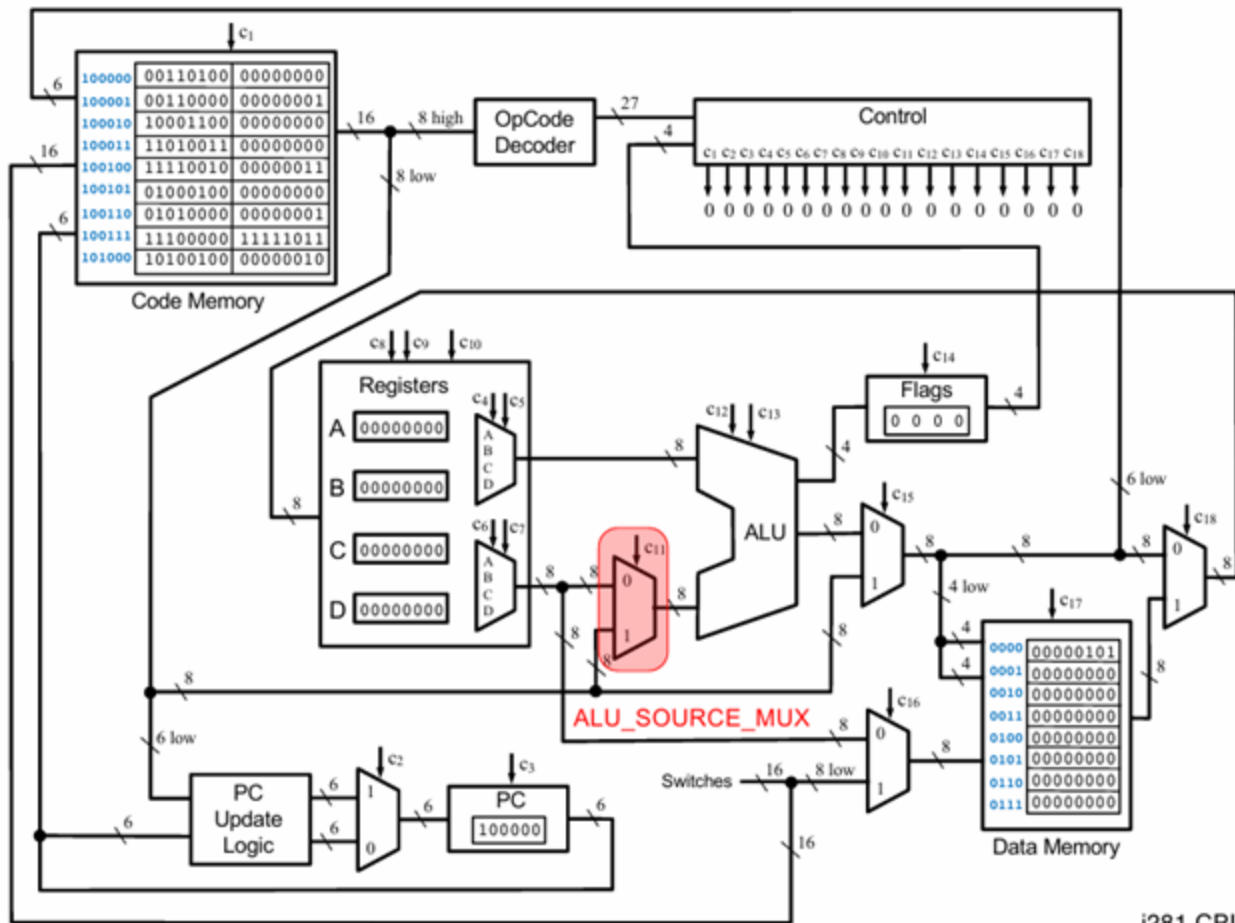
Zero Flag (ZF)
Negative Flag (NF)
Overflow Flag (OF)



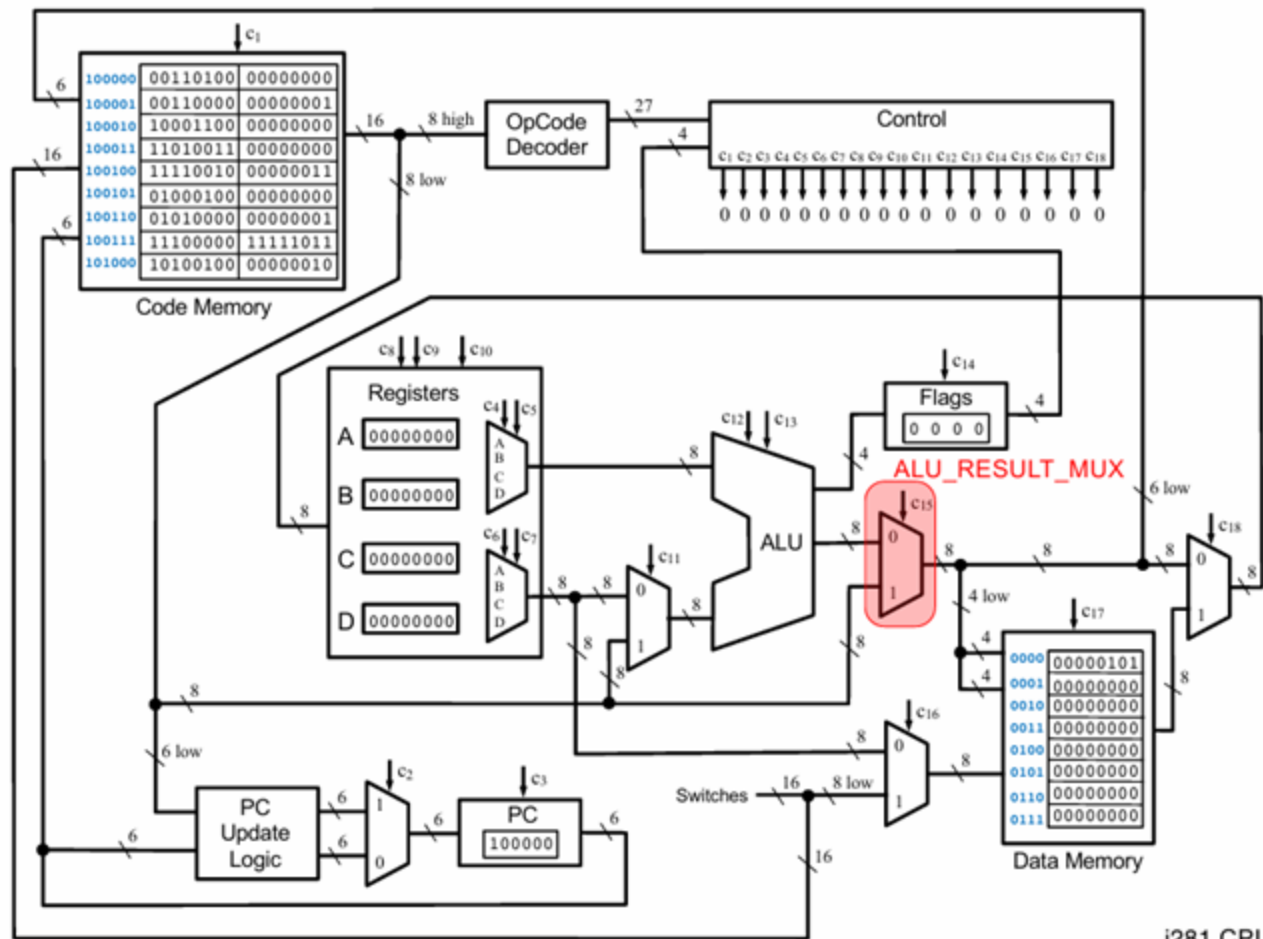
i281 CPU

2-to-1 Bus Multiplexer (with 8-bit lines)

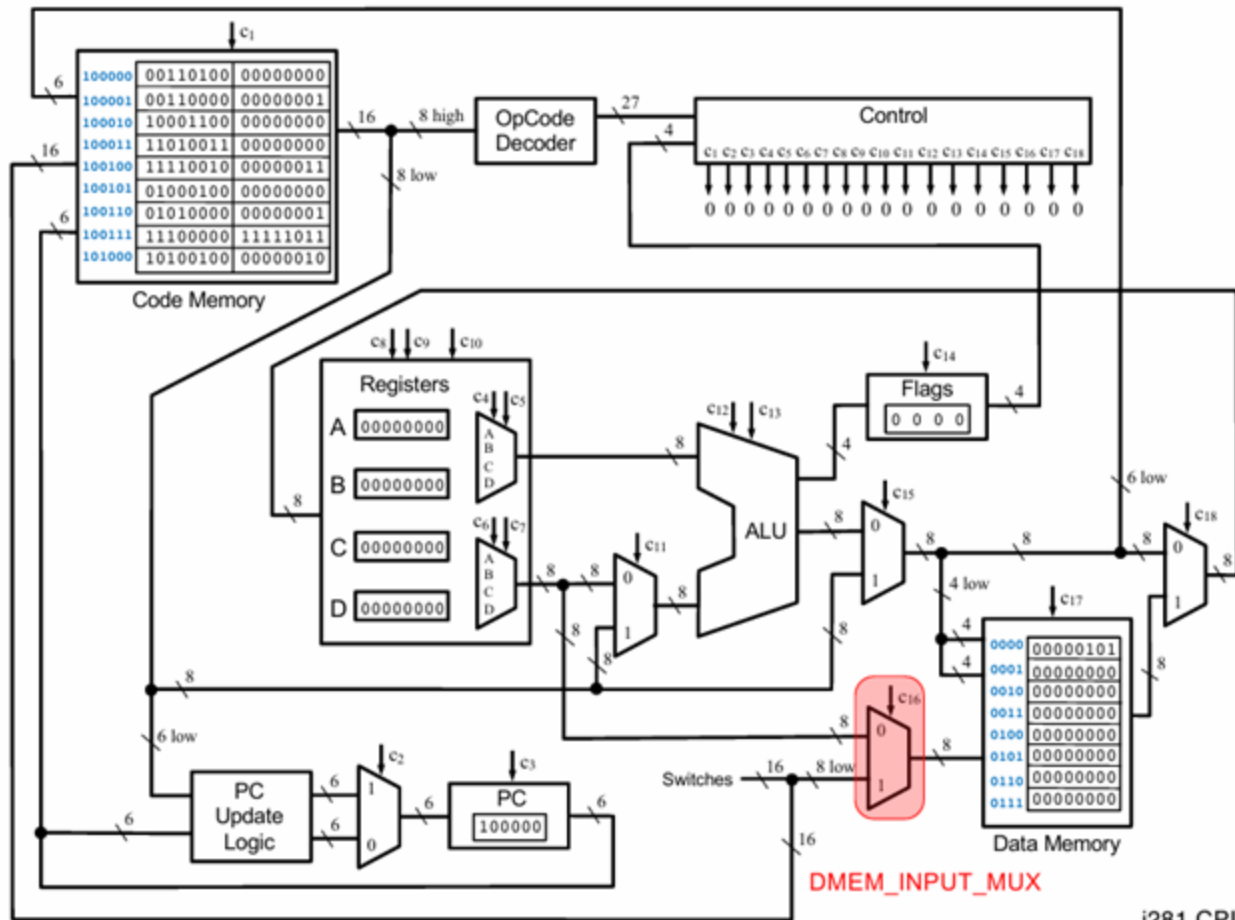




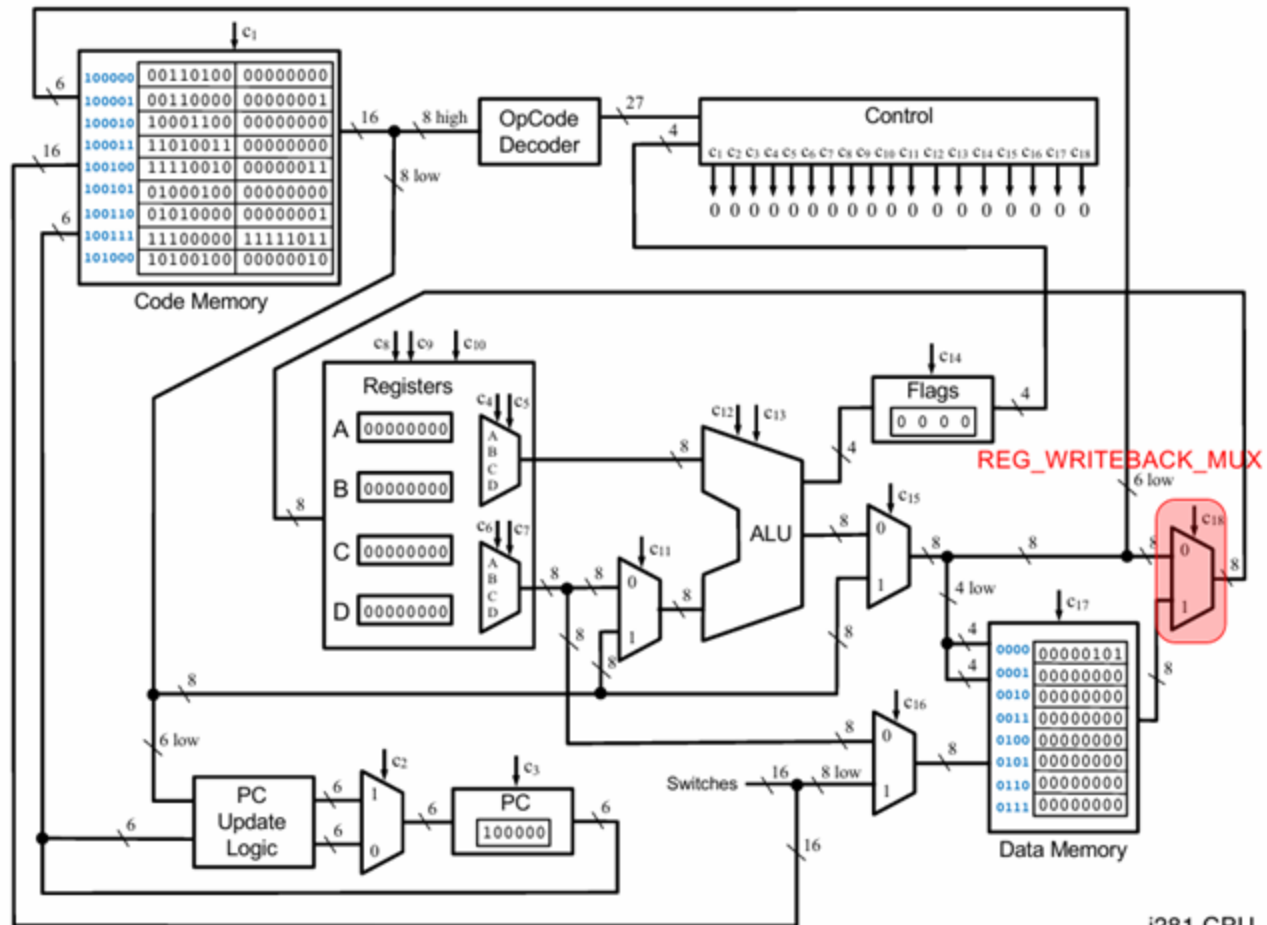
i281 CPU



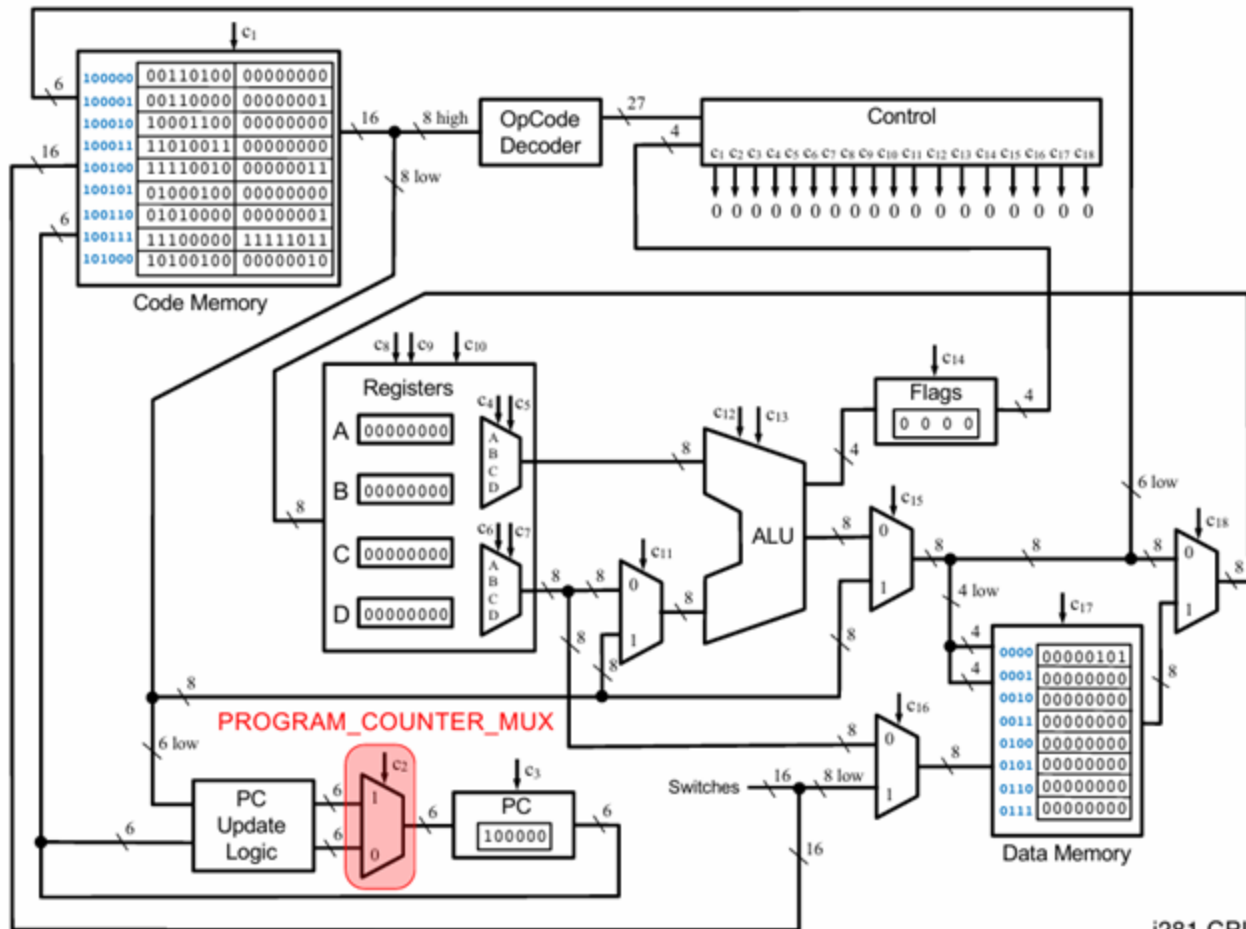
i281 CPU



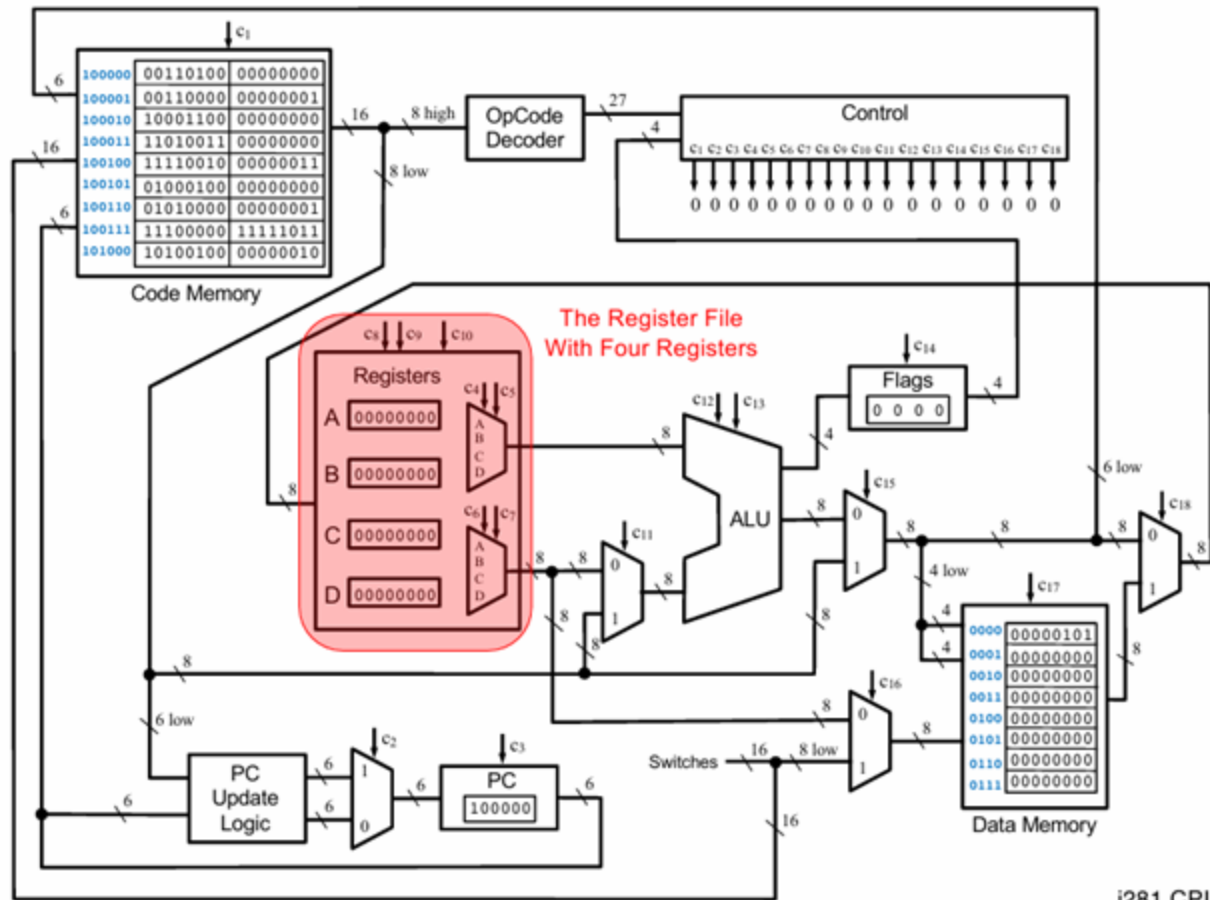
i281 CPU



i281 CPU

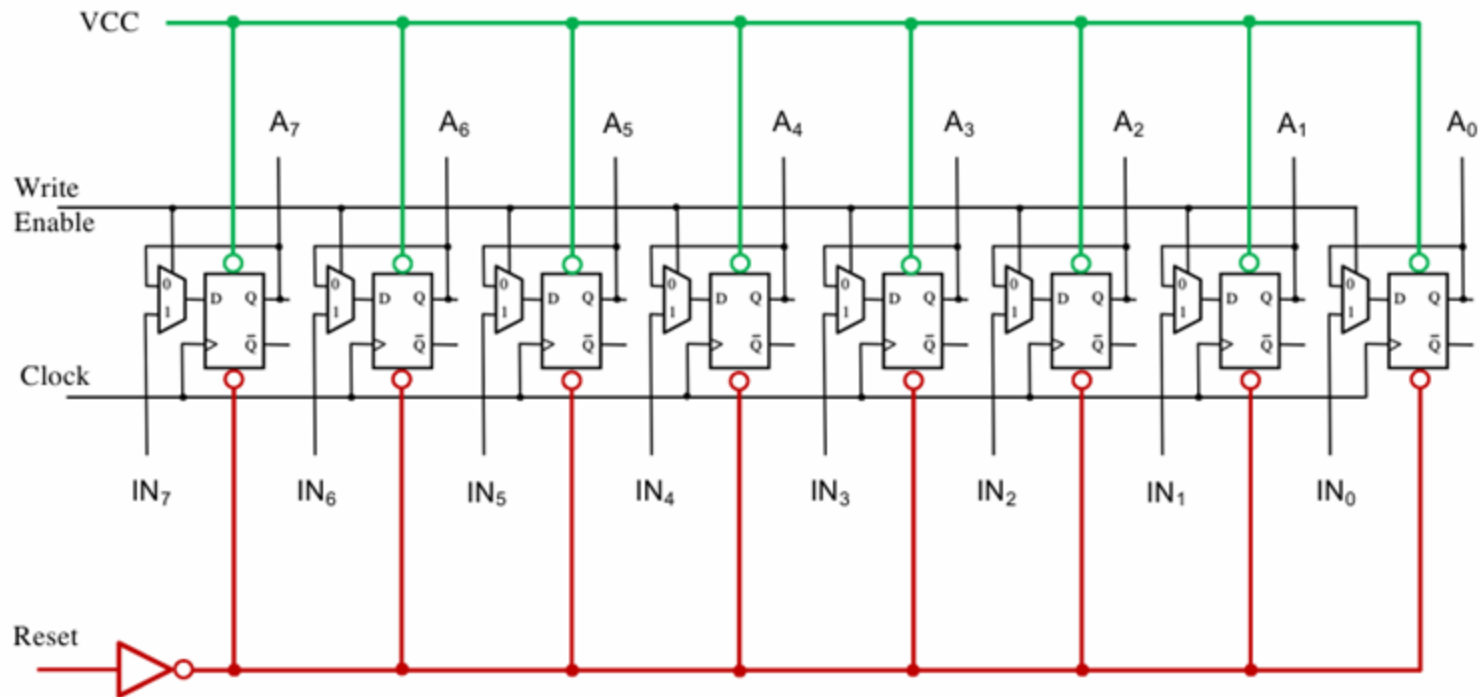


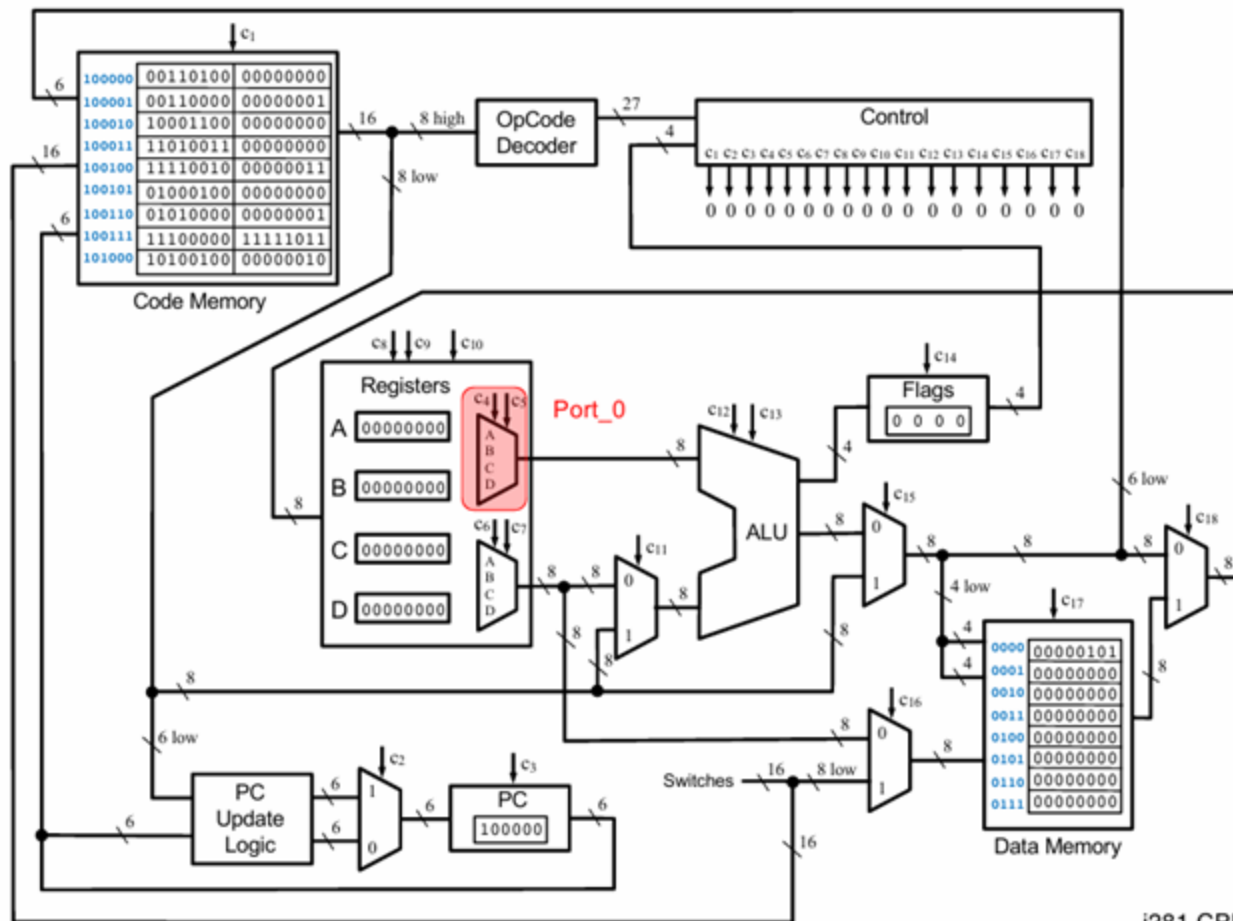
i281 CPU



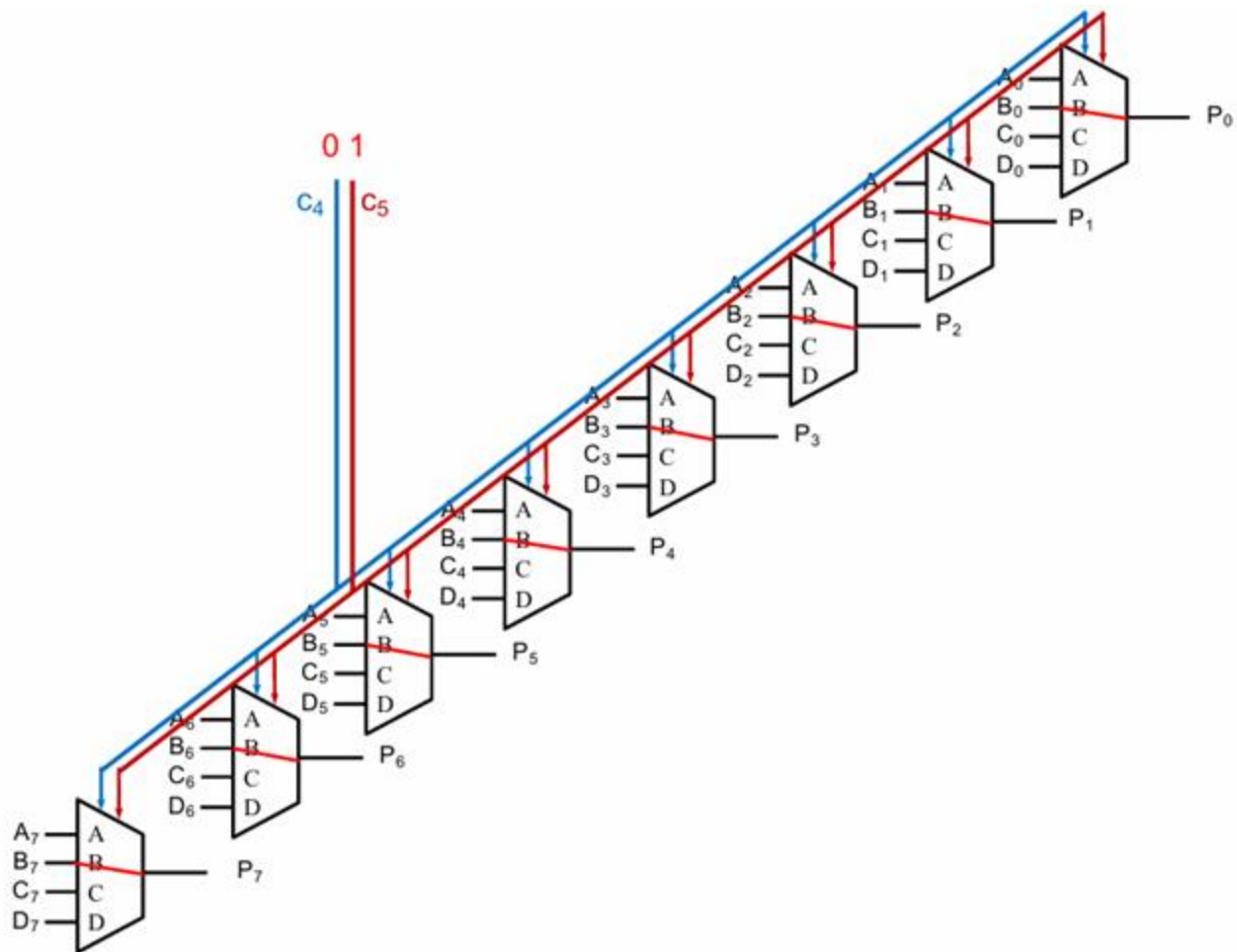
i281 CPU

Register A





i281 CPU



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
```

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);
}
```

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 ?

Data Memory:

05
00
00

.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:

34 00
30 01
8C 00
D3 00
F2 03
44 00
50 01
E0 FB
A4 02

Assembly Language

Machine Language
in Hexadecimal

OPCODE Mapping

.data

N	BYTE	5
i	BYTE	?
sum	BYTE	?

Data Memory:

00000101
00000000
00000000

.code

Code Memory:

	LOADI	B, 0	0011_01_00_00000000
	LOADI	A, 1	0011_00_00_00000001
	LOAD	D, [N]	1000_11_00_00000000
Loop:	CMP	A, D	1101_00_11_00000000
	BRG	End	1111_00_10_00000011
Add:	ADD	B, A	0100_01_00_00000000
	ADDI	A, 1	0101_00_00_00000001
	JUMP	Loop	1110_00_00_11111011
End:	STORE	[sum], B	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

Code Memory:

	LOADI	B, 0	0011_01_00_00000000
	LOADI	A, 1	0011_00_00_00000001
	LOAD	D, [N]	1000_11_00_00000000
Loop:	CMP	A, D	1101_00_11_00000000
	BRG	End	1111_00_10_00000011
Add:	ADD	B, A	0100_01_00_00000000
	ADDI	A, 1	0101_00_00_00000001
	JUMP	Loop	1110_00_00_11111011
End:	STORE	[sum], B	1010_01_00_00000010

The OPCODEs

NOOP

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

SHIFTR

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

CMP

1	1	0	1	R	X	R	Y	d	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

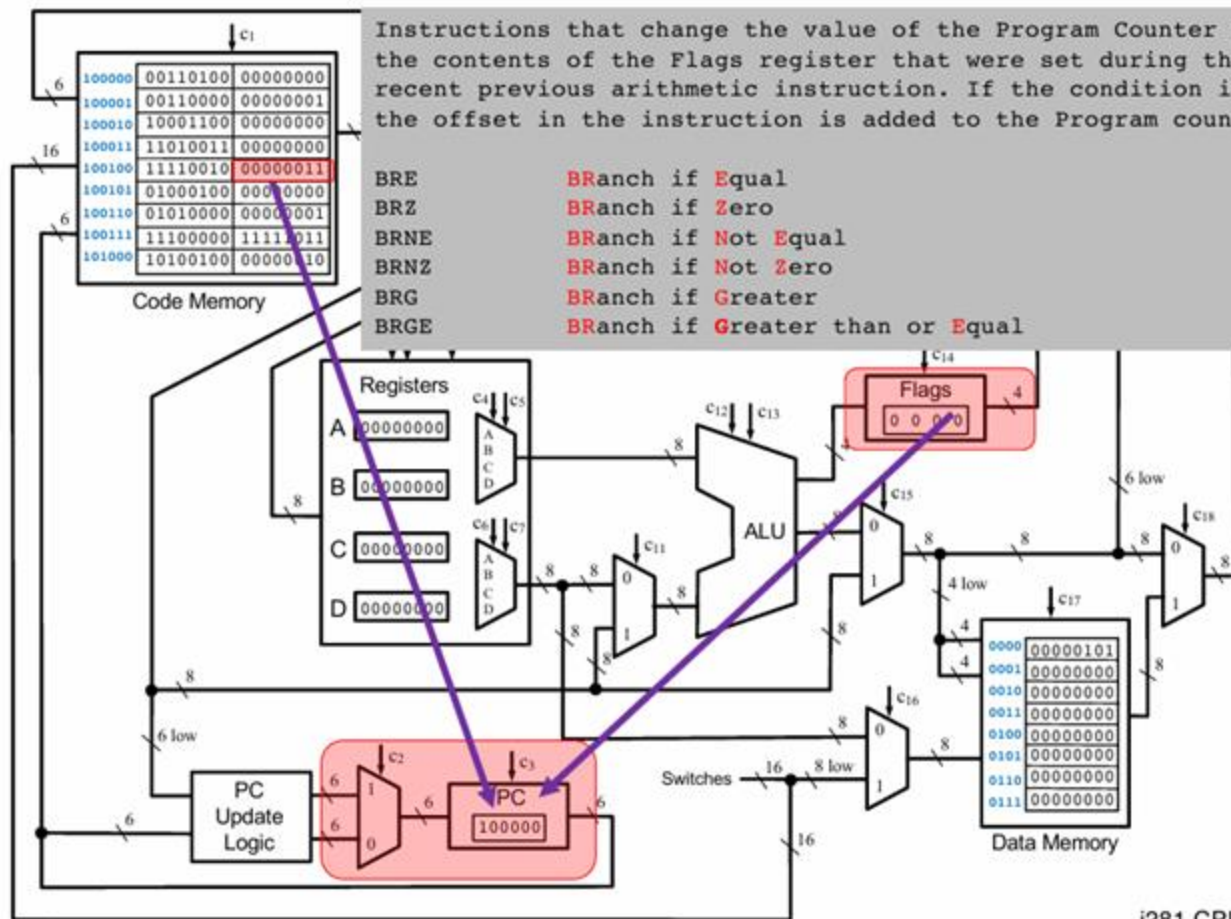
1	1	1	1	d	d	1	0	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

BRGE

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

Instructions that change the value of the Program Counter based on the contents of the Flags register that were set during the most recent previous arithmetic instruction. If the condition is met the offset in the instruction is added to the Program counter.

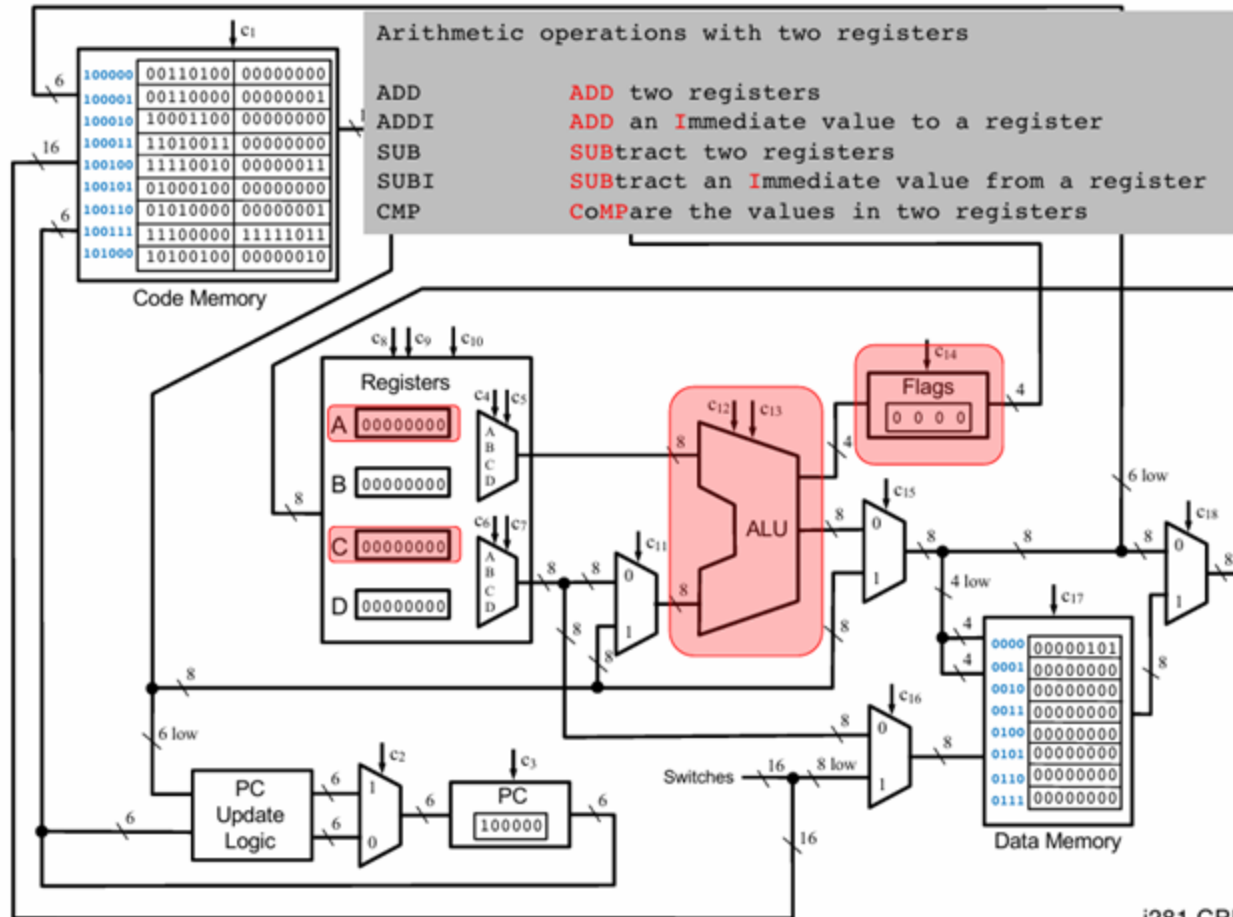
BRE BRanch if Equal
 BRZ BRanch if Zero
 BRNE BRanch if Not Equal
 BRNZ BRanch if Not Zero
 BRG BRanch if Greater
 BRGE BRanch if Greater than or Equal

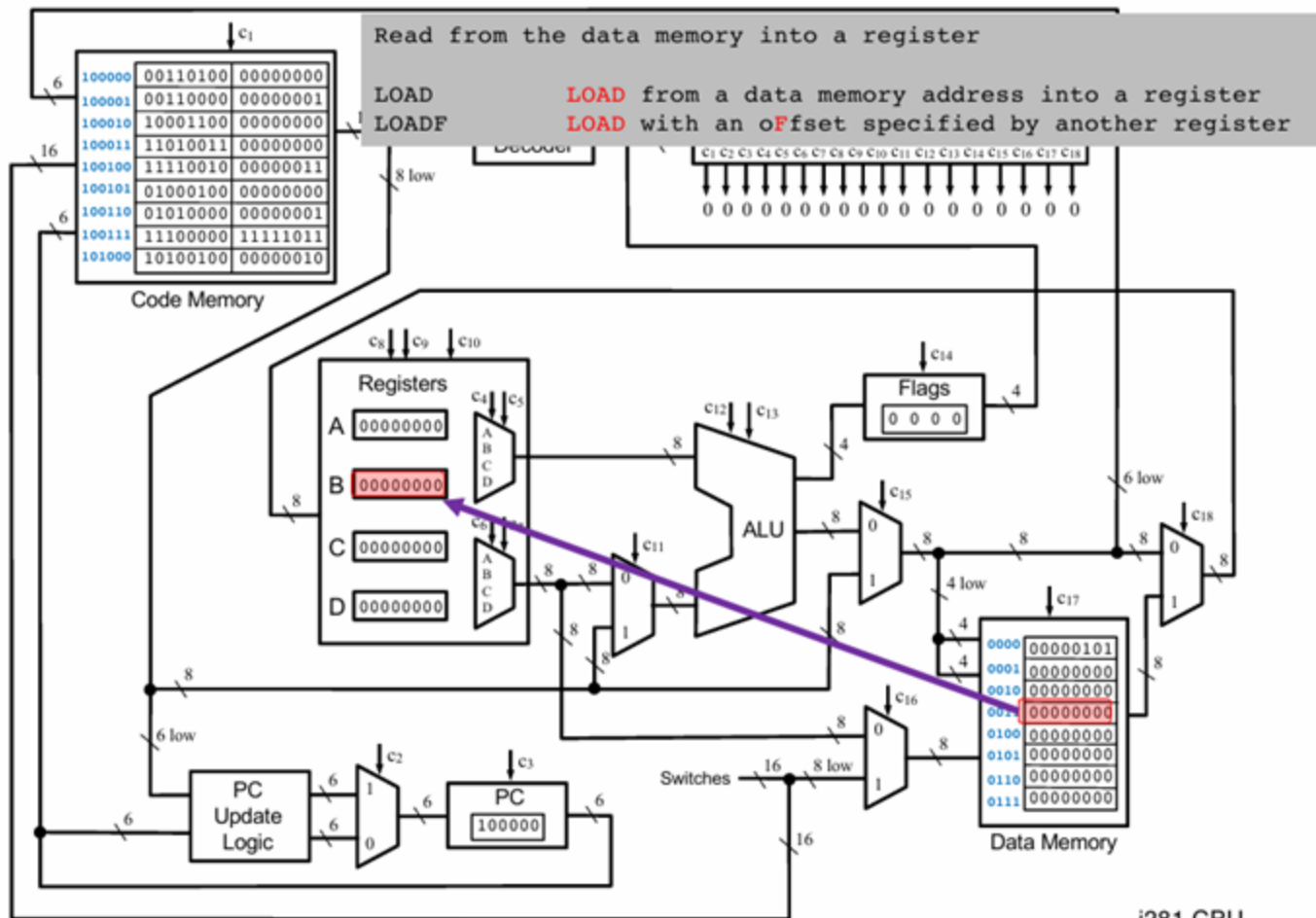


i281 CPU

Arithmetic operations with two registers

ADD **A**DD two registers
 ADDI **A**DD an **I**mmEDIATE value to a register
 SUB **S**UBtract two registers
 SUBI **S**UBtract an **I**mmEDIATE value from a register
 CMP **C**OMPare the values in two registers



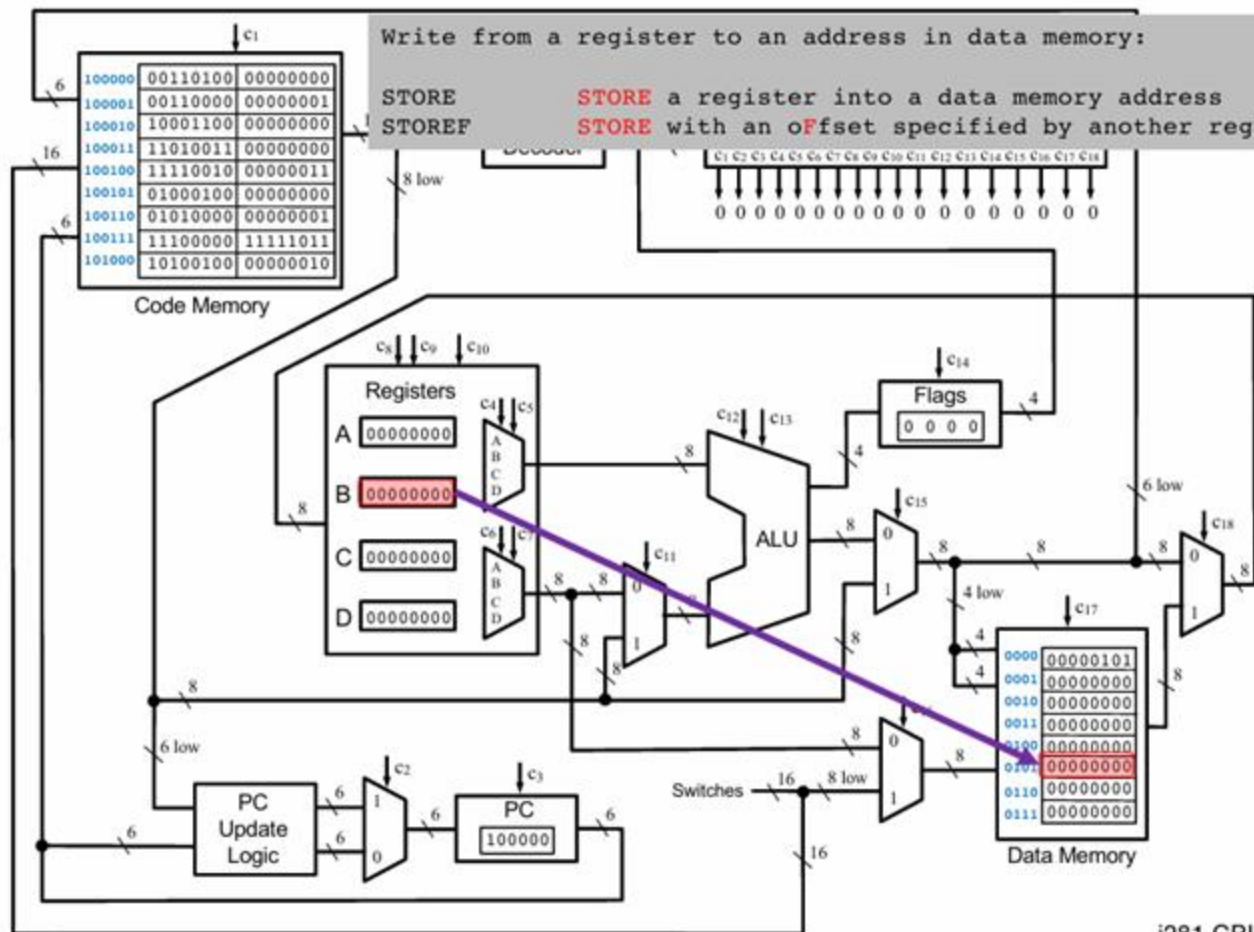


i281 CPU

Write from a register to an address in data memory:

STORE **STORE** a register into a data memory address

STOREF **STORE** with an **offset** specified by another register



There are 16 possible ADD instructions

ADD A, A

0	1	0	0	0	0	0	0	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ADD A, B

0	1	0	0	0	0	0	1	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ADD A, C

0	1	0	0	0	0	1	0	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ADD A, D

0	1	0	0	0	0	1	1	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ADD B, A

0	1	0	0	0	1	0	0	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ADD B, B

0	1	0	0	0	1	0	1	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ADD B, C

0	1	0	0	0	1	1	0	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ADD B, D

0	1	0	0	0	1	1	1	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

There are 16 possible ADD instructions

ADD C, A

0	1	0	0	1	0	0	0	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ADD C, B

0	1	0	0	1	0	0	1	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ADD C, C

0	1	0	0	1	0	1	0	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ADD C, D

0	1	0	0	1	0	1	1	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ADD D, A

0	1	0	0	1	1	0	0	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ADD D, B

0	1	0	0	1	1	0	1	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

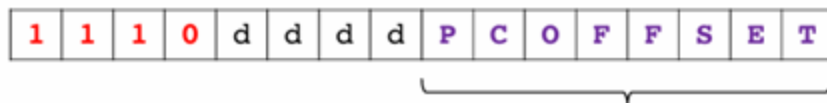
ADD D, C

0	1	0	0	1	1	1	0	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ADD D, D

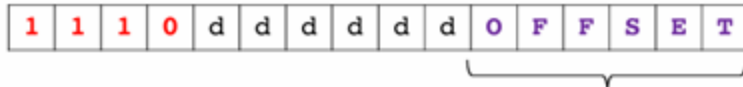
0	1	0	0	1	1	1	1	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

JUMP Label



These 8 bits specify an offset for the program counter (PC).
The offset is encoded in 2's complement representation
and can be either positive or negative.

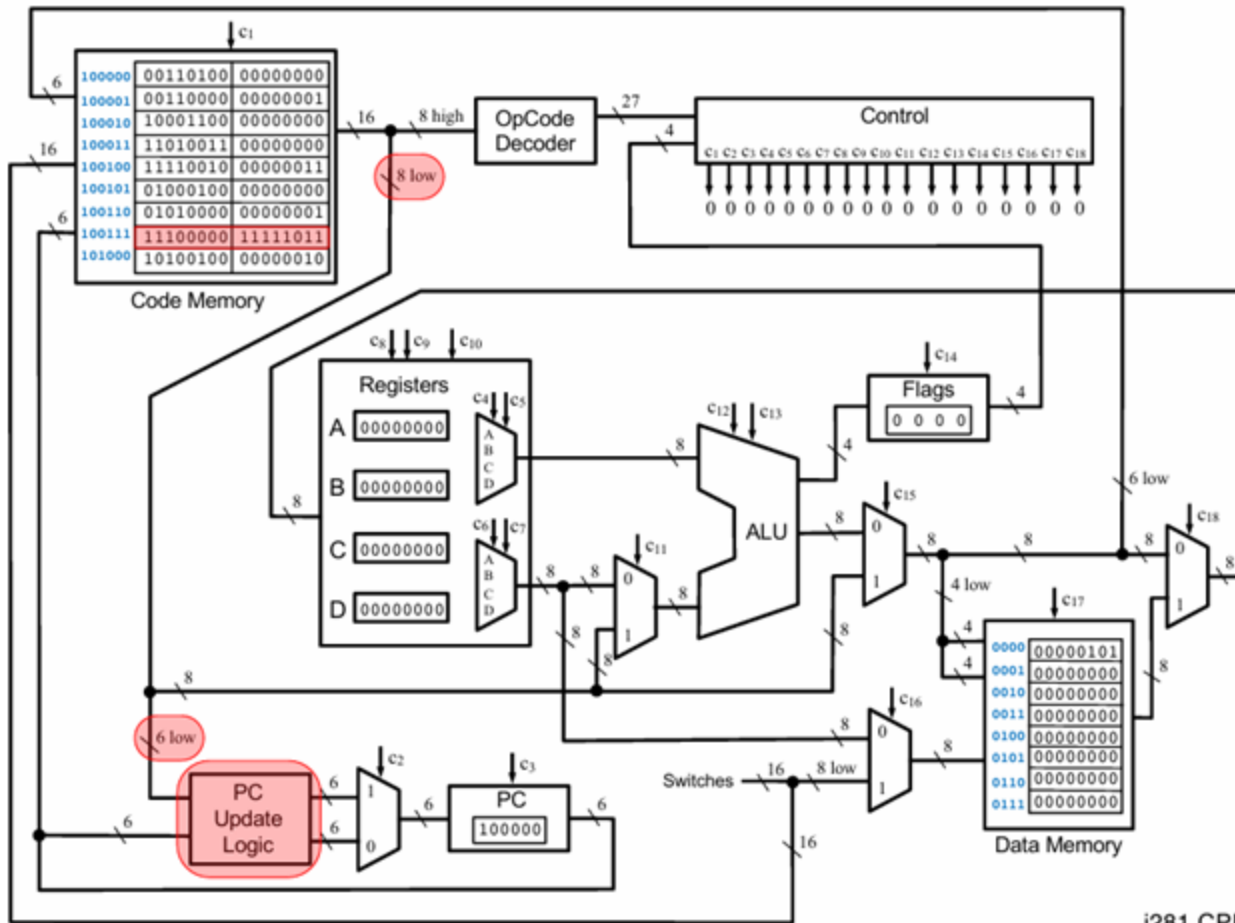
JUMP Label



Because the code memory of the i281 CPU has space for only
64 instructions, however, the hardware uses only the last 6 bits.
That is, it uses 6-bit adders to compute addresses.

Therefore, the possible range of offset values ranges from -32 to +31.
However, due to an implicit +1 offset implemented by the hardware
in the PC update logic, the actual effective range is -31 to +32.

The compiler emits 6-bit numbers that are then sign extended to 8-bit.



i281 CPU

