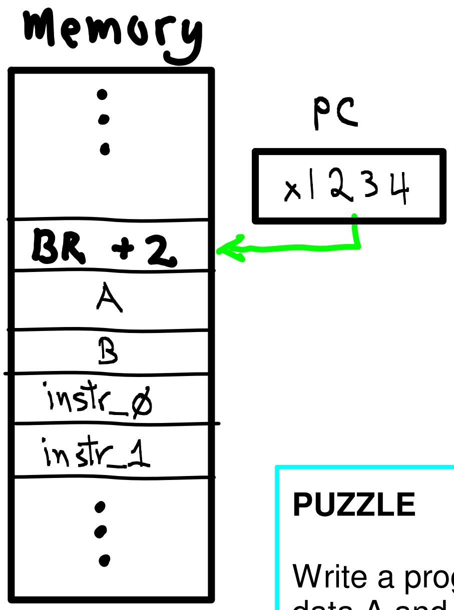
LC3 ISA, micro-Architecture, and Programming



LC3 Memory

address space: 16'd0 — 16'hFFFF 16-bit **MAR**

Word size: 16-bits, 16-bit **MDR**

addressability: word (16-bit) (2-Byte)

memory word size = MAR size = MDR size

Not generally True for all machines

PUZZLE

Write a program in LC3 machine code which alters its first two instructions using data A and B:

(1) **instr_0.**opcode <== **instr_0.**opcode + **A** (only opcode is altered)
(2) **instr_1.**opcode <== **instr_1.**opcode + **B** (likewise)

regardless of where the program might be located in memory when executing. Assume PC initially contains address of memory word above A. Data A and B are in consecutive memory words as part of program. Don't worry about what happens after the end of the program is reached.

Below, we will show LC3 states with, (1) Register Transfer Language (**RTL**) indicating the operation, and (2) the required **NON-ZERO control signals**. For example,

MAR <== **PC**
LD_MAR

indicates that **PC** content transfers into **MAR**, and **LD_MAR** control signal must be 1 (all other control signals are assumed to be 0.) Necessary signal paths are shown like this, for example,

IR[15:12]--->**FSM**.in

indicates that the 4 high-order bits of the **IR** need to be routed to the control **FSM**'s input.

The test bench, "top_rtl_testInstr", in the test.jelib Electric library displays the current simulation tick, the FSM's state, all non-zero control signals, and all non-zero MUX controls, eg.,

```
-------------------------------------( 3 )--------------------------
-------((( 18 )))-------[ LD_MAR ]---------[ ]-----------------
```

Here, the current tick is 3, the state is 18, the LD_MAR == 1, and all MUX selects are zeroes. Next we would also see values for the PC, MAR, MDR, IR, PSR, and all eight registers in the RegFile.

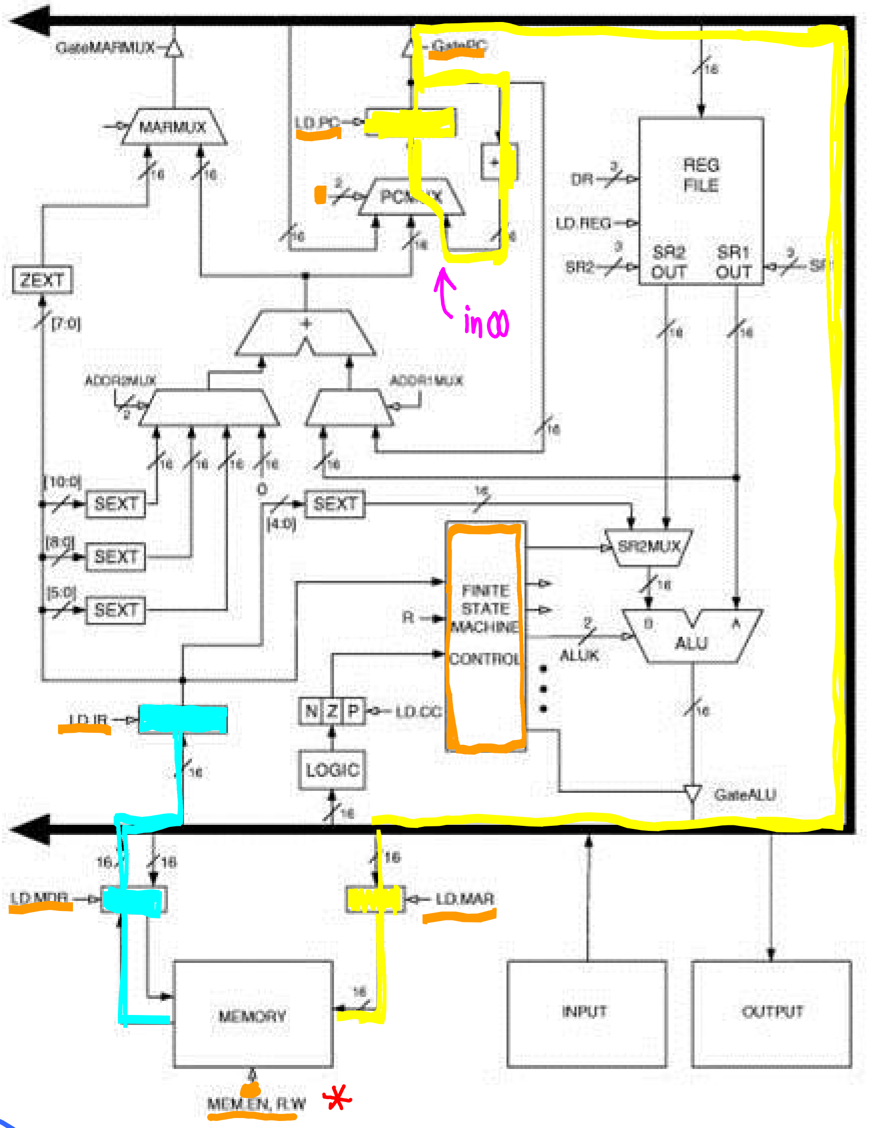
fetch instruction:

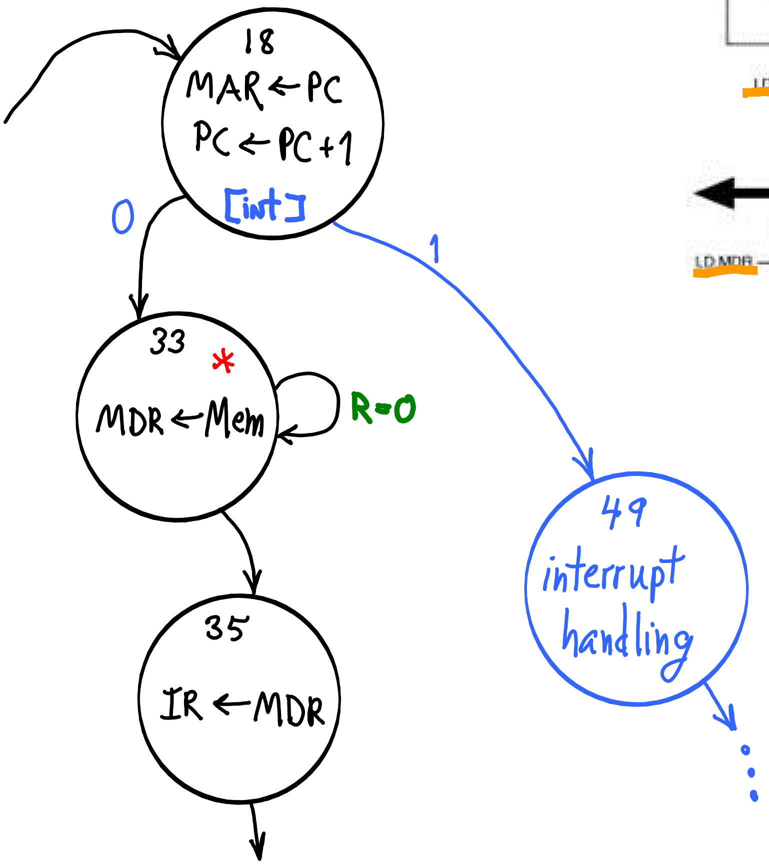
State 18:
  MAR <== **PC**
    GatePC
    LD_MAR

  PC <== **PC** + 1
    PCMUX = 00 (select)
    LD_PC

State 33:
  MDR <== **MEM**.out
    LD_MDR
    MIO_EN *
    R_W = 0

State 35:
  IR <== **MDR**
    LD_IR *





To 32, Decode

(See App. C)

Branch on **int**:
  [int]
Branch on **R**:
  "R=0"

* See Tri-states in MEMORY-IO BUS

mio bus:
- databus
- address bus
- control bus

in Electric

1. See **top.Mem-IO-Bus**

   --- address decode
   --- tri-states
   --- control bus

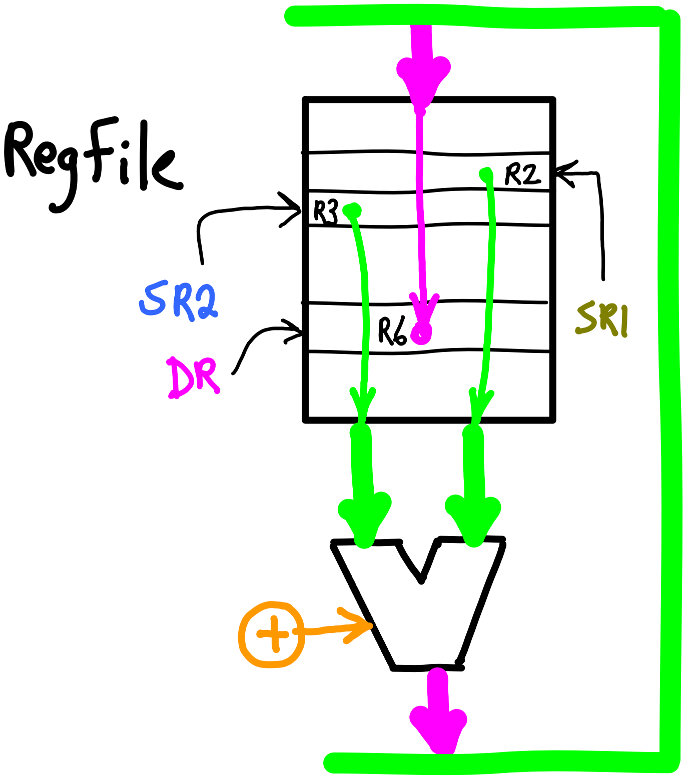
2. See **test.testInstr**

   --- initializing memory

Instruction Formats

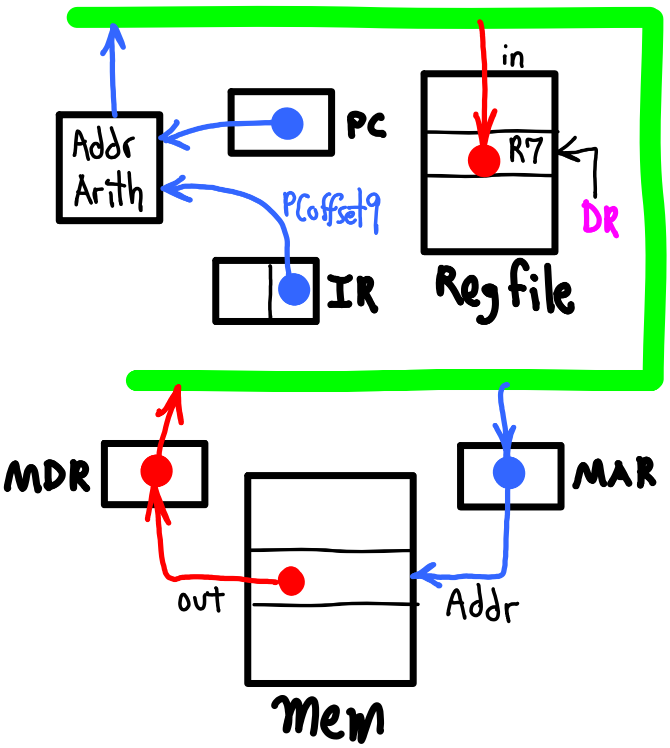
	Opcode	DR	SR1	...	SR2
bits:	15...12	11...9	8...6	5...3	2...0
	0001	110	010	000	011
	ADD	R6	R2		R3

$$R6 \leftarrow R2 + R3$$



	Opcode	DR/SR1	PCoffset9
bits:	15...12	11...9	8......0
	0010	111	0 0001 0110
	LD	R7	x16

$$R7 \leftarrow \text{Mem}[PC + \text{PCoffset9}]$$



	Opcode	DR/SR1	BaseR	offset6
bits:	15...12	11...9	8...6	5...0
	0110	111	011	01 1000
	LDR	R7	R3	#12

$$R7 \leftarrow \text{Mem}[\,R3 + \text{offset6}\,]$$

} Same as LD, but uses Reg instead of PC

Operate Instructions (ADD, AND, NOT)

State-**9** (**NOT**):

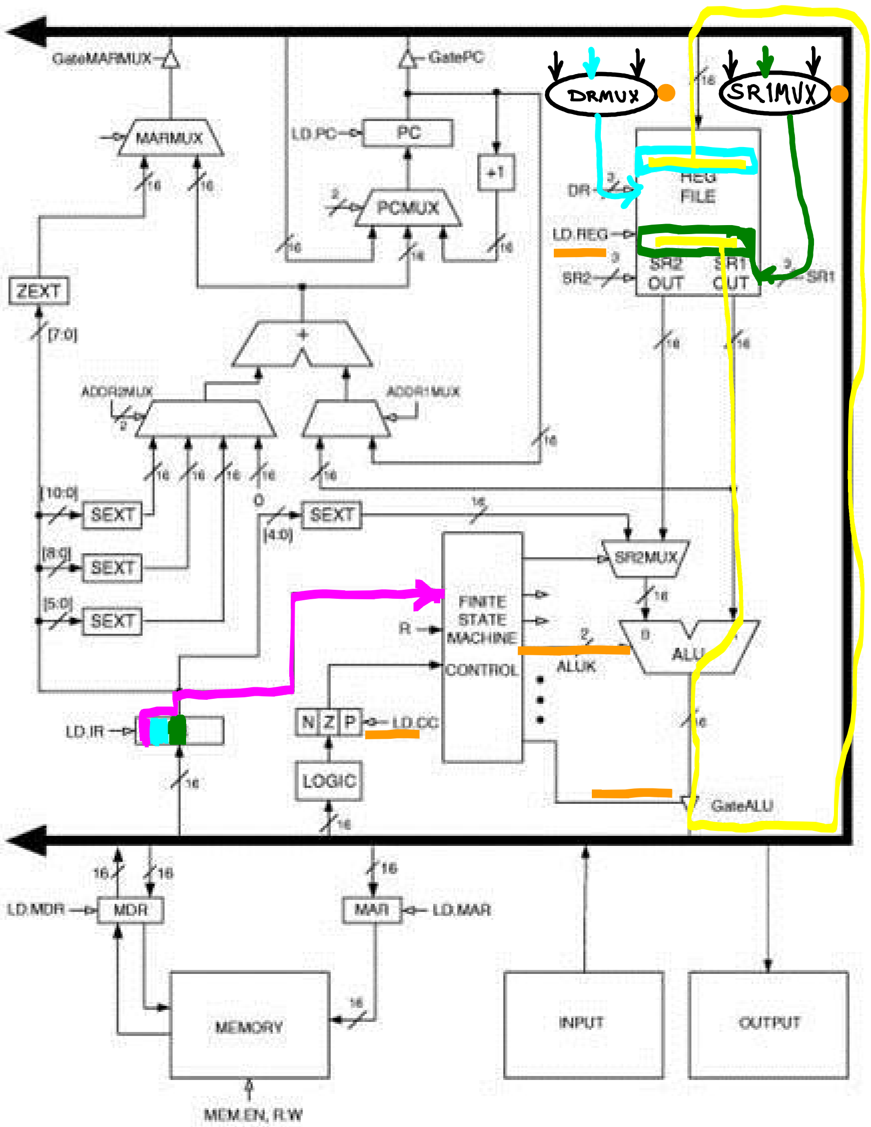
IR[15..12] ---> **FSM**.in
IR[11..9] ---> **DRMUX** ---> **RegFile**.DR
IR[8..6] ---> **DRMUX** ---> **RegFile**.SR1
ALU.out ---> **RegFile**.in

DR <== **NOT**(**SR1**)
GateALU SR1MUX.select == ?
LD_REG DRMUX.select == ?
LD_CC ALUK = 10 (NOT)

R3 ← NOT(R5)

	NoT	DR	SR1	
	1001	011	101	...
bits:	15...12	11...9	8...6	5......0

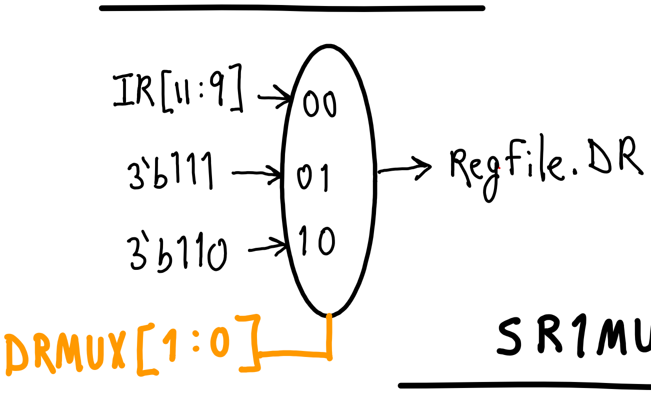
Register-Register Addressing



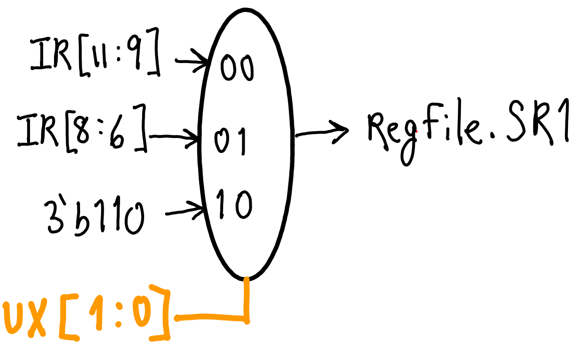
MUX'ed RegFile INPUTS (see, App. C, p. 574)

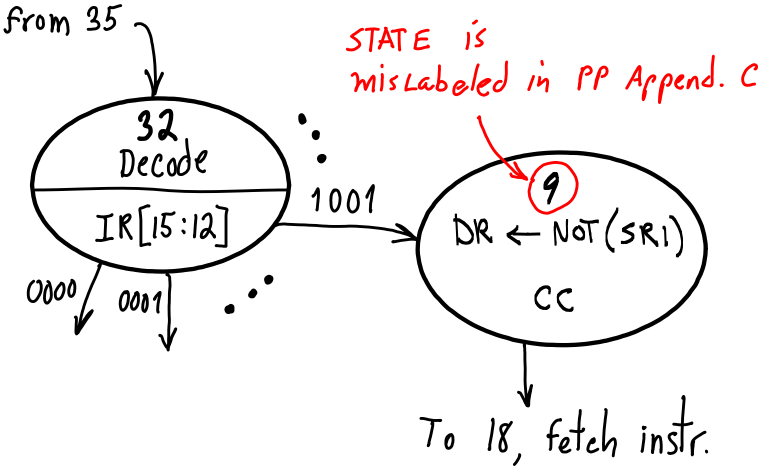
Control Signals: DRMUX[1 : 0]
SR1MUX[1 : 0]

DRMUX



SR1MUX





Before:

Regfile[101] = 1100101011110000

After:

Regfile[011] = 0011010100001111

Regfile[101] = 1100101011110000

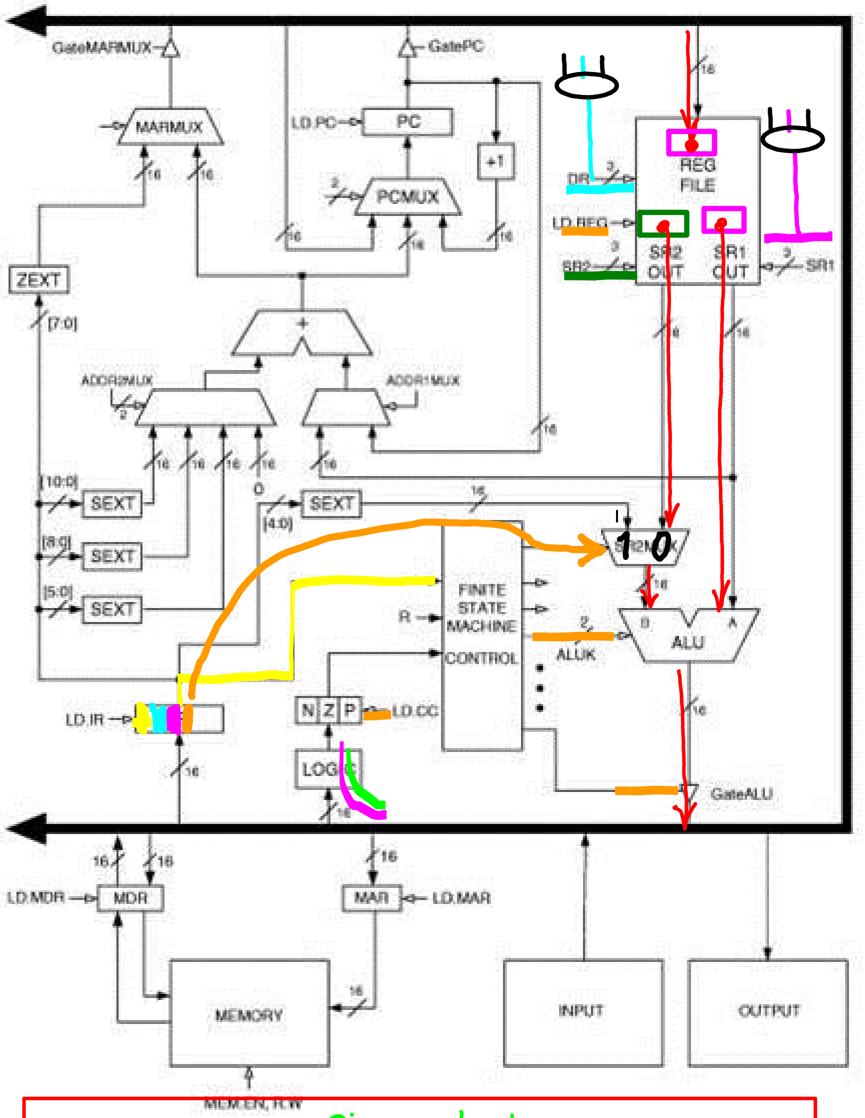
ADD (3-register addressing)

State-**1**:

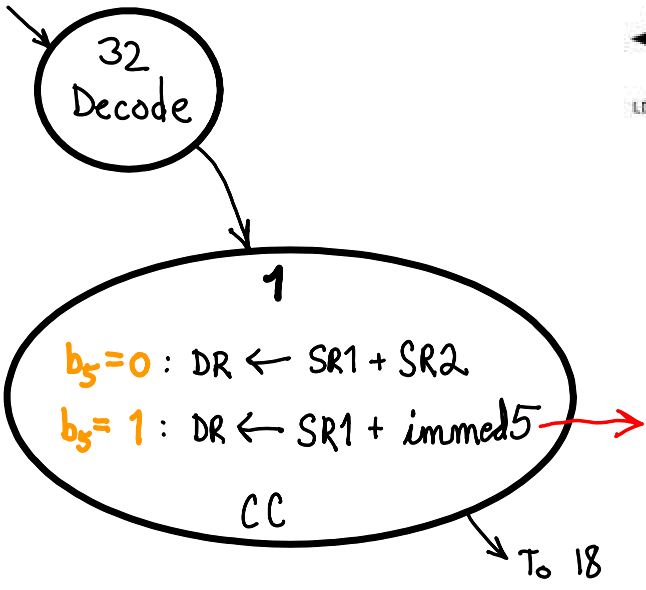
- **IR**[15..12] **---> FSM**.in
- **IR**[11..9] **---> DRMUX ---> RegFile**.DR
- **IR**[8..6] **---> SR2MUX ---> RegFile**.SR1
- **IR**[2..0] **---> RegFile**.SR2
- **IR**[5] **---> SR2MUX**

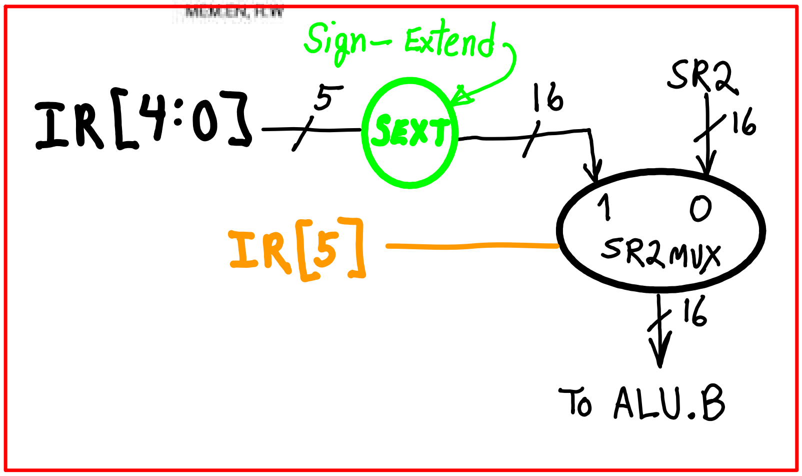
DR <=== RegFile[SR1] + RegFile[SR2]

GateALU
LD_REG
LD_CC
ALUK = 00



	ADD	DR	SR1		SR2
	0001	011	100	0...	101
bits:	15...12	11...9	8...6	5	2...0



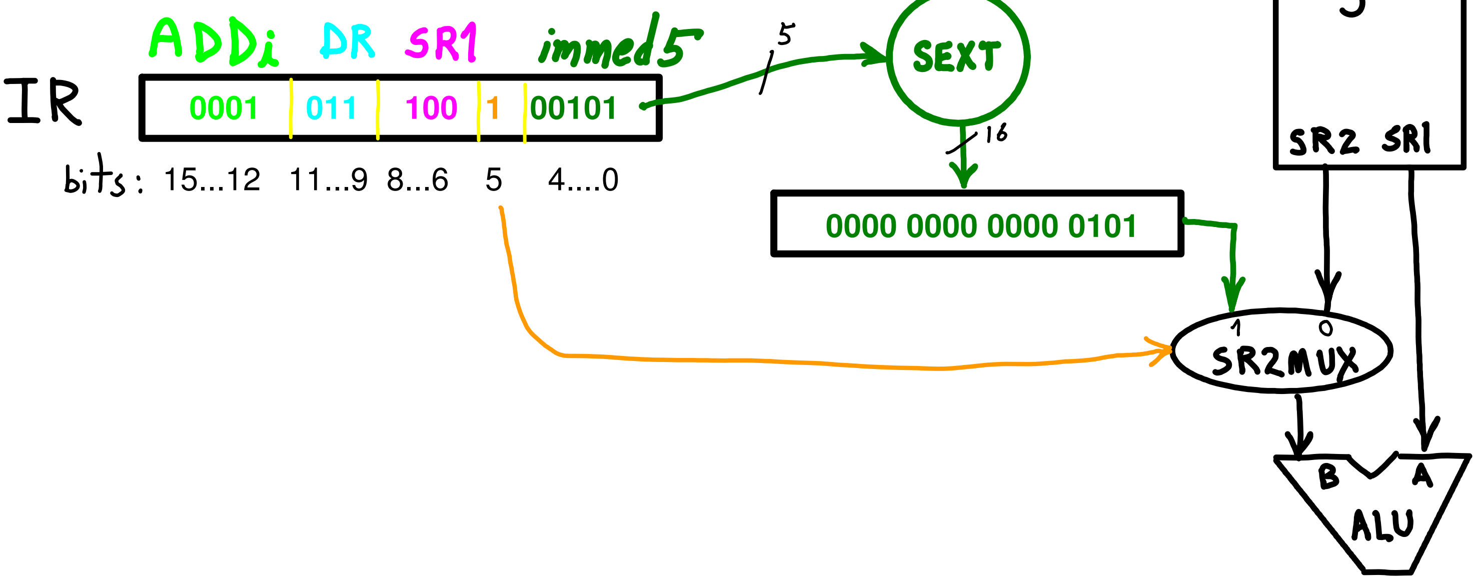


ADD can function two ways:

1. **ADD**: Get both operands from **RegFile,** SR1 and SR2
   ---- ***register-register-register addressing***

2. **ADDi**: Get one operand from **RegFile** (SR1) and the other from **IR**[4 : 0]
   ---- **IR**[4 : 0] (aka, ***immed5*** in this context) is sign-extended from a 5-bit number to a 16-bit number.
   ---- Sign-extending copies the low 5 bits, and then makes the upper 11 bits all 0 or all 1, depending on whether ***immed5*** is positive or negative
   ---- ***register-register-immediate addressing***

Sign-extending the **IR** immediate data bits:
IR[4 : 0] **--->** **SEXT**.in **--->** **ALU**.B



A - B ? ===> Do **A + (-B)**

2s-complement with immediate constants.

Suppose: **A** in **R0**, **B** in **R1**

1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 1

NOT R1 R1

0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1

ADD R1 R1 immed5

$$\left.\begin{array}{l} R1 \leftarrow NOT(R1) \\ \\ R1 \leftarrow R1 + 1 \end{array}\right\} R1 \leftarrow (-R1)$$

0 0 0 1 0 1 1 0 0 0 0 0 1 0

ADD R2 R0 R1

$R2 \leftarrow R0 + R1$

$$R2 \leftarrow \underset{A}{R0} - \underset{B}{R1}$$

(But, R1 now has -B)

Load/Store (LD / ST ; pc-relative addressing **)**
load a register from memory / store register in memory
DOES Address ARITHMETIC

LD
State-**2**:

IR[8..0] **---> SEXT-9x16 ---> ADDR2MUX**
PC ---> ADDR1MUX
(**ADDR2MUX + ADDR1MUX**) **---> MARMUX**

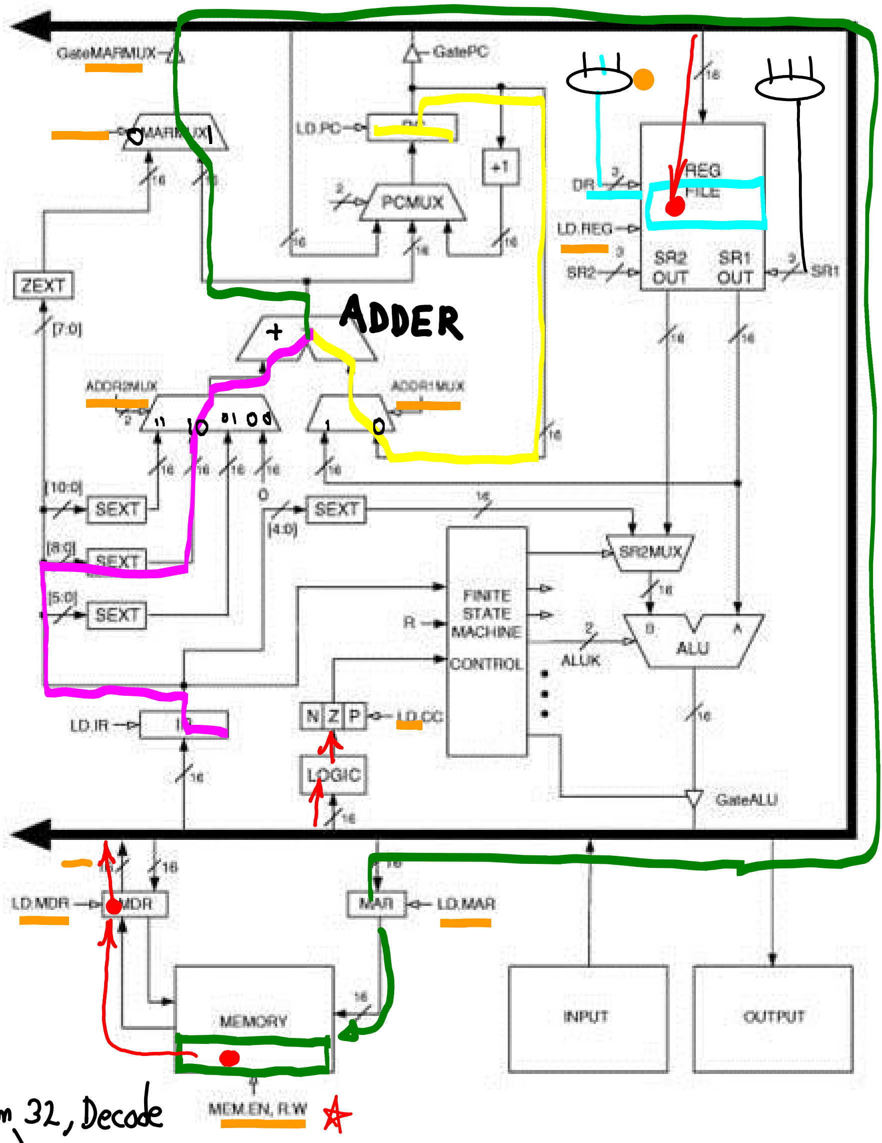
MAR <=== PC + IR[8..0]
GateMARMUX ADDR1MUX == 1'b0
LD_MAR ADDR2MUX == 2'b10

State-**25**:

MDR <=== MEM.out
LD_MDR
MIO_EN
R_W == 0 } ☆ + Tri-states on MIO BUS

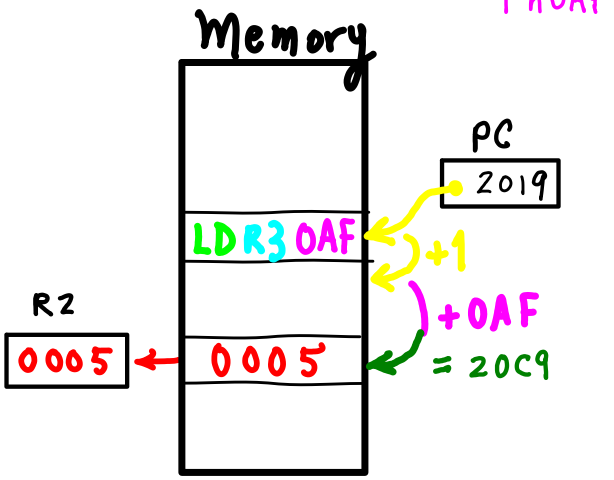
State-**27**:

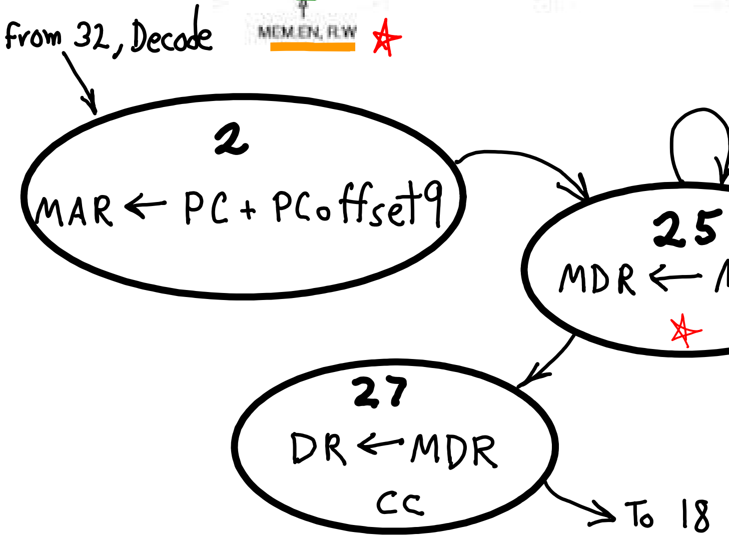
DR <=== MDR
GateMDR LD_REG
LD_CC DRMUX == ?



	LD	DR	PCoffset9
	0010	011	0 1010 1111
bits:	15...12	11...9	8......0

x 0AF
9`h0AF





PC	<==	**PC**+1	**0010 0000 0001 1010**	**(x201A)**
		+ **SEXT**(**IR**[8 : 0 **]**)	+ **0000 0000 1010 1111**	**(x00AF)**
MAR	<==		= **0010 0000 1100 1001**	**(x20C9)**

R2 <== **MDR** <== **MEM[x20C9]** (**x0005**)

LDI / STI

(memory indirect addressing)

Pointers Variables

LDI:

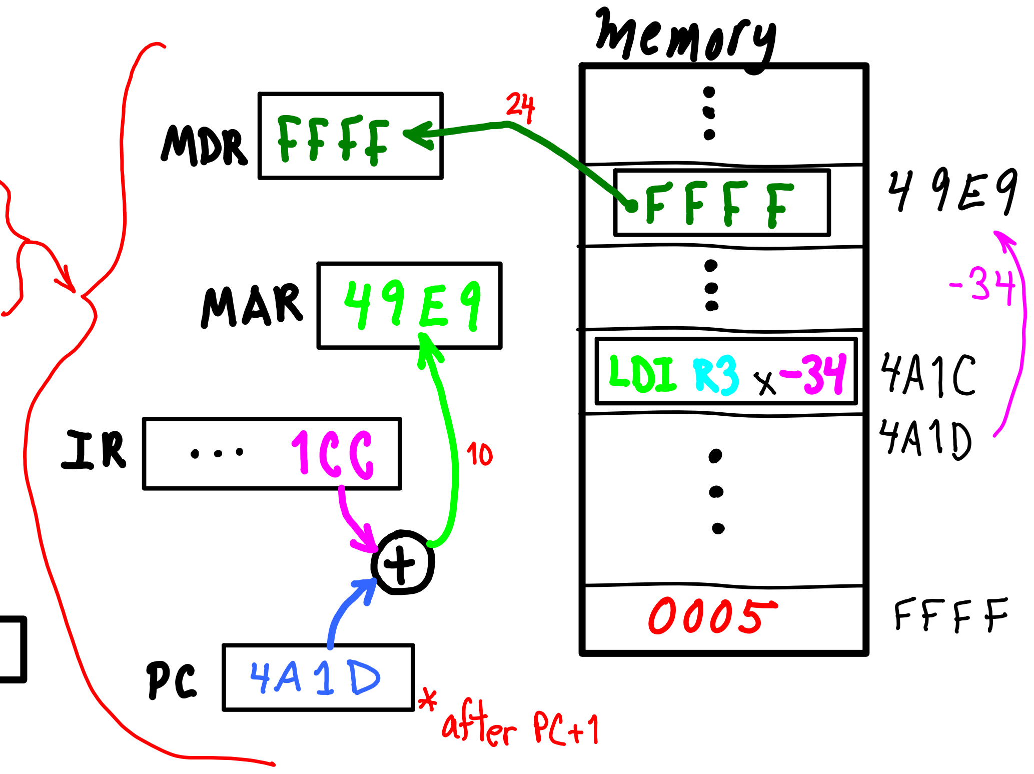
State-**10**:
    MAR <=== PC + IR[8..0]

State-**24**:
    MDR <=== MEM[PC + PCoffset9]

State-**26**:
    MAR <=== MDR

State-**25**:
    MDR <=== MEM[MAR]

State-**27**:
    DR <=== MDR



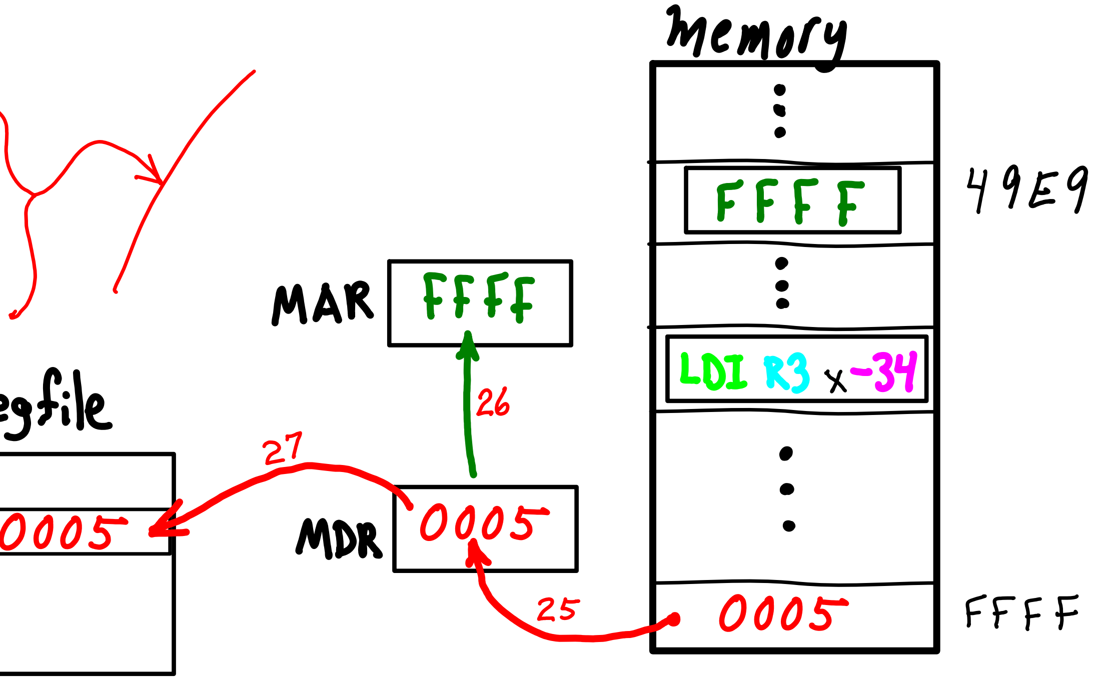
LDI	R3	PCoffset9
1010	011	1 1100 1100
bits: 15...12	11...9	8......0

x1CC (-x34)

State-**26**:
    MAR <=== MDR

State-**25**:
    MDR <=== MEM[MAR]

State-**27**:
    DR <=== MDR



Sanity check

```
 1 1
 4 A 1 D
+ F F C C
---------
1 4 9 E 9
```

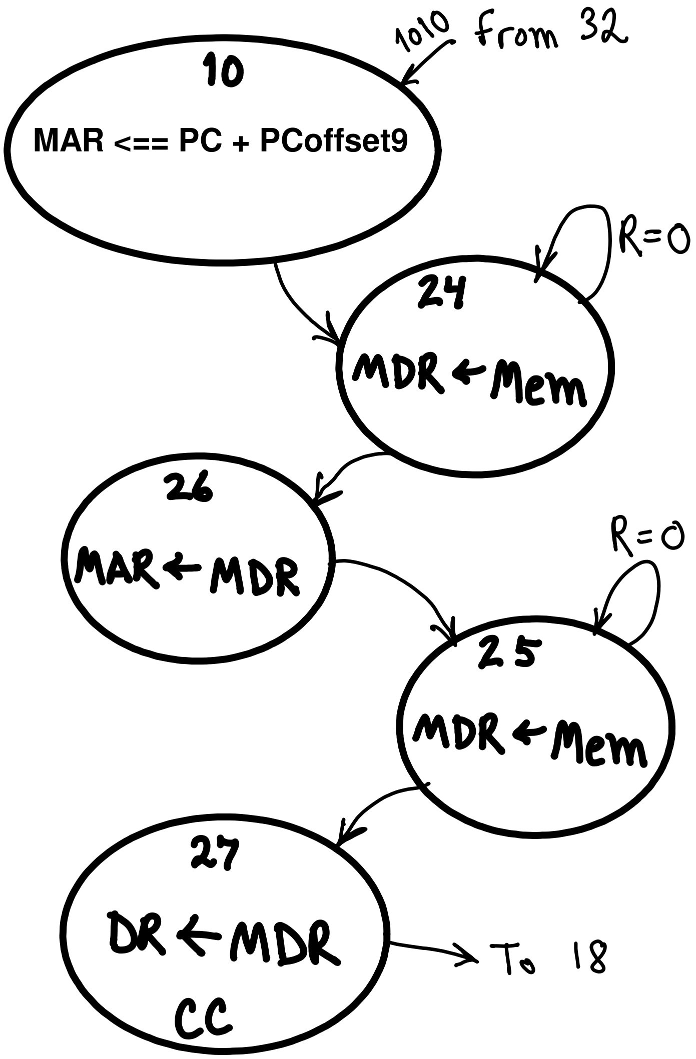
```
    1
 4=0100
+F=1111
-------
1 0011
```

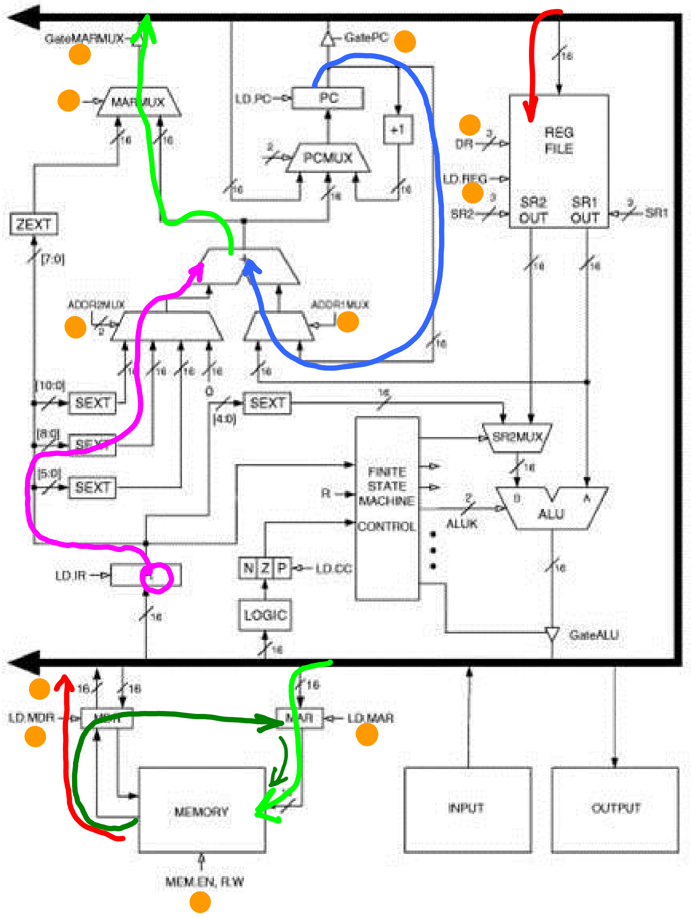
```
   1 1
 A=1010
+F=1111
-------
1 1001
```

```
       1
 D = 1101
+C = 1100
---------
1   1001
```

A	1010
B	1011
C	1100
D	1101
E	1110

```
  -1
 4A1D
 - 34
-----
 49E9
```





What we've got so far:

NOT R1, R1

ADD R1, R2, R3
ADDi R1, R2, x10

AND R1, R2, R3
ANDi R1, R2, #13

5-bit data ⇒ ±16 $(2^5 = 32)$ $[-16, +15] \subseteq \mathbb{Z}$

decimal

LD R1, R2, x-34
ST R1, R2, x-34

LDI R1, R2, x-34
STI R1, R2, x-34

9-bit PC offset ⇒ ±258 $(2^9 = 512)$

hex

More addressing range — 16-bit

LDR / STR (register-indirect addressing)
Pointer in a register

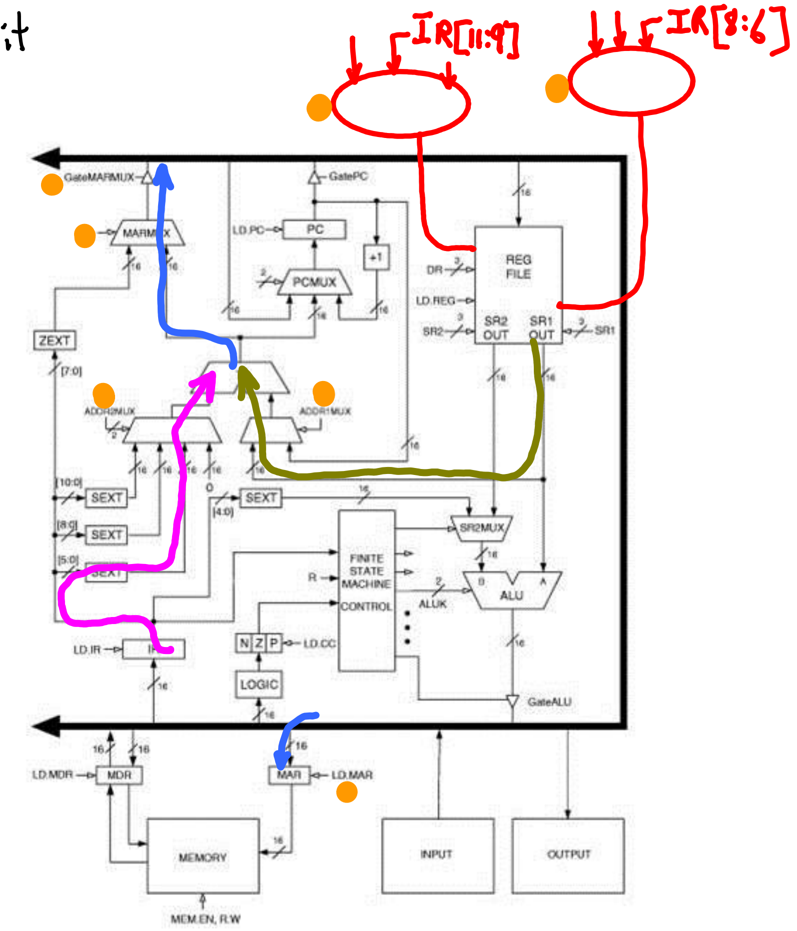
LDR

State-6:
- IR[11..9] ---> RegFile.DR
- IR[8..6] ---> RegFile.SR1
- RegFile.SR1out ---> ADDR1MUX
- IR[5..0] ---> ADDR2MUX

MAR <== BaseR + offset6

State-25:
MDR <== MEM[MDR]

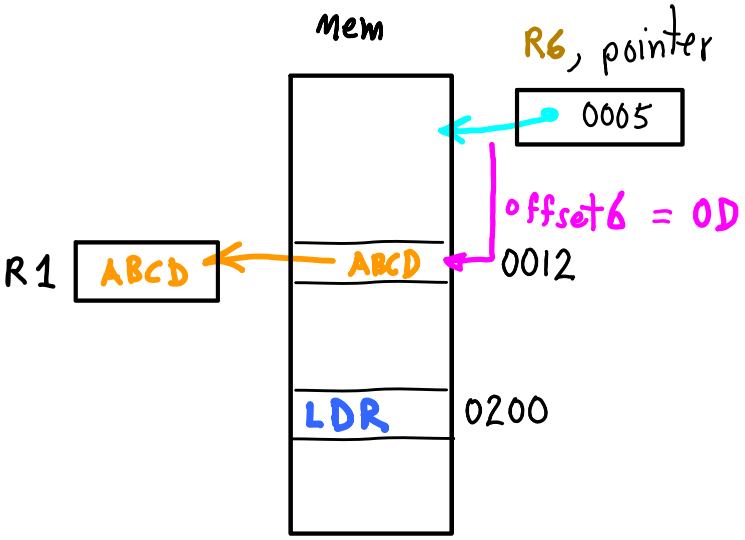
State-27:
DR <== MDR



	LDR	R3	R6	offset6
	0110	011	110	001101
bits:	15...12	11...9	8...6	5......0

full 16-bit addressing

address	Memory content
...	
x0012:	ABCD
...	
x0200:	1010 011 110 001101
...	



Overall Effect

DR <=== MEM[BaseR + Offset6]

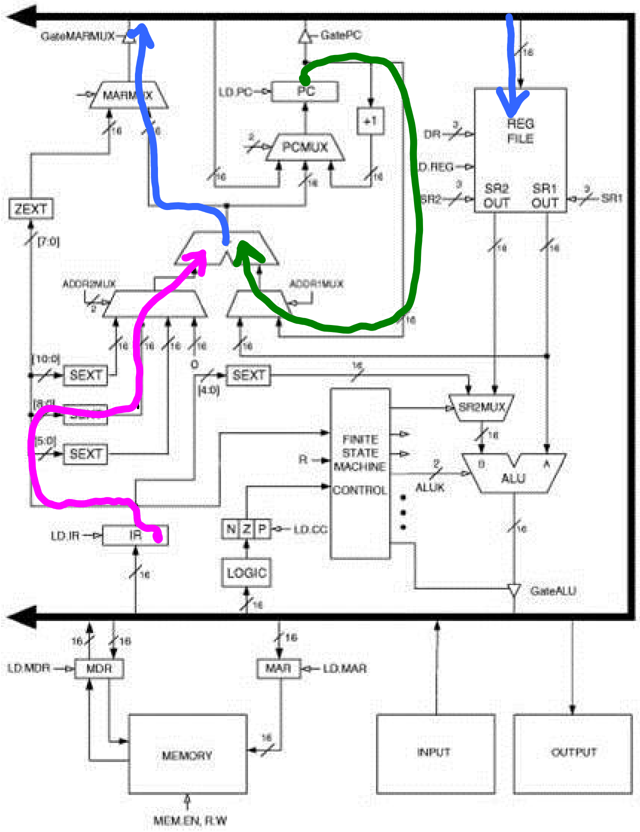
But, how did we get an address (16-bit) into R6?

LEA
(immediate addressing)

State-**14**:

PC ---> **ADDR1MUX**
IR[8..0] ---> **ADDR2MUX**
MARMUX ---> **RegFile**.in

DR <= **PC** + **PCoffset9**

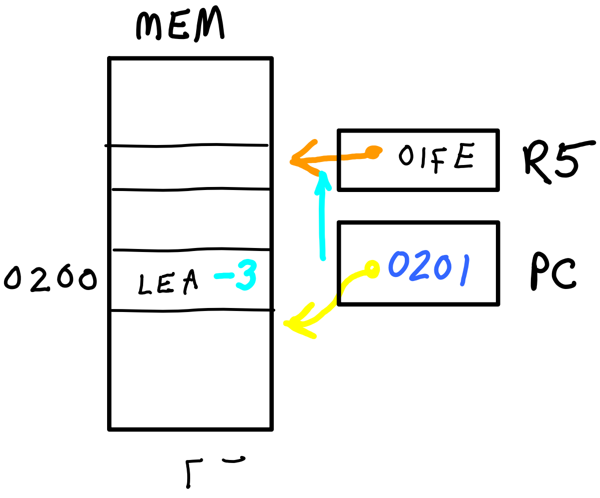


	LEA	R3	offset9
	1110	011	1 1111 1101
bits:	15...12	11...9	8......0

1FD (-3)

Memory Content

x01FE:
x01FF:
x0200: 1110 101 111111101 (**PC** <== x0201)
x0201:



R5 <=== **PC** + **SEXT**(**1FD**)

1111 1111 1111 1101
0000 0010 0000 0001

0201 + FFFD = 0201 - 3 = 01FE