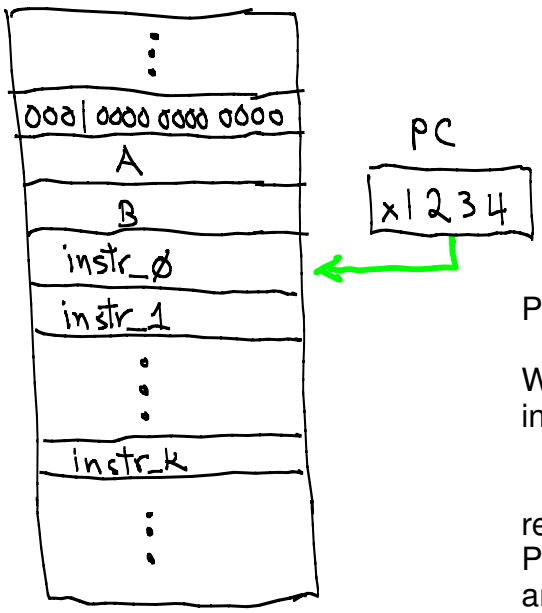
LC3 ISA, micro-Architecture, and Programming

But first, a puzzle.



Memory (LC3)

address space: $0-2^{16}$, 16-bit MAR
word size: 16-bits, 16-bit MDR
addressability: word (16-bit) (2-Byte)
NB - memory word size = MAR size = MDR size
but 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 <== opcode + A, only opcode is altered.
(2) instr_1: opcode <== opcode + B, likewise.

regardless of where the program might be located in memory. Assume PC initially contains address of memory word containing instr_0. 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 explore LC3 instructions and their execution. Register Transfer Language (RTL) indicates the operation and the required control signals are listed. For example,

```
MAR <== PC
    LD_MAR
```

indicates that the content of PC 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
```

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

NB--Control state numbers look strange: F1 is state-18, F2 is state-33, F3 is state-35, etc.

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

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

indicates the current tick is 3, the current state is 18, the LD_MAR is 1, and all MUX select signals are zeroes. Following the above is a listing of the content of all CPU registers (PC, MAR, MDR, IR, PSR, and all eight registers in RegFile).

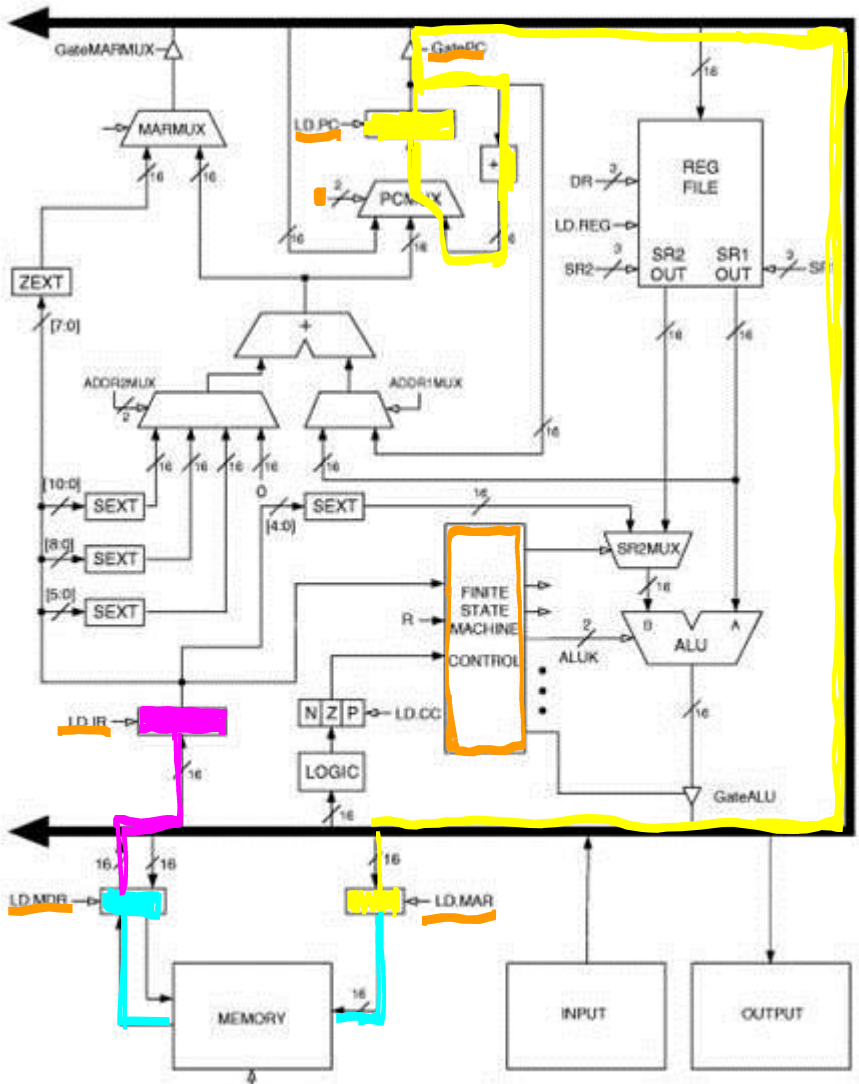
fetch instruction:

State 18:
MAR <= PC,
GatePC
LD_MAR

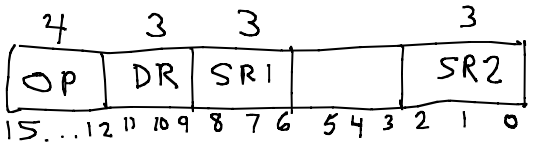
PC <= PC + 1
PCMUX = 00 (select)
LD_PC

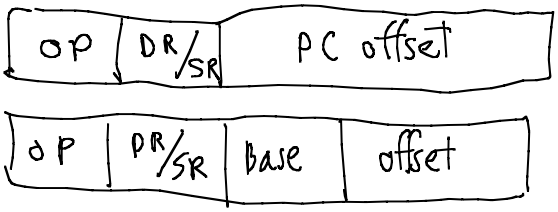
State 33:
MDR <= MEM_OUT
LD_MDR

State 35:
IR <= MDR
LD_IR

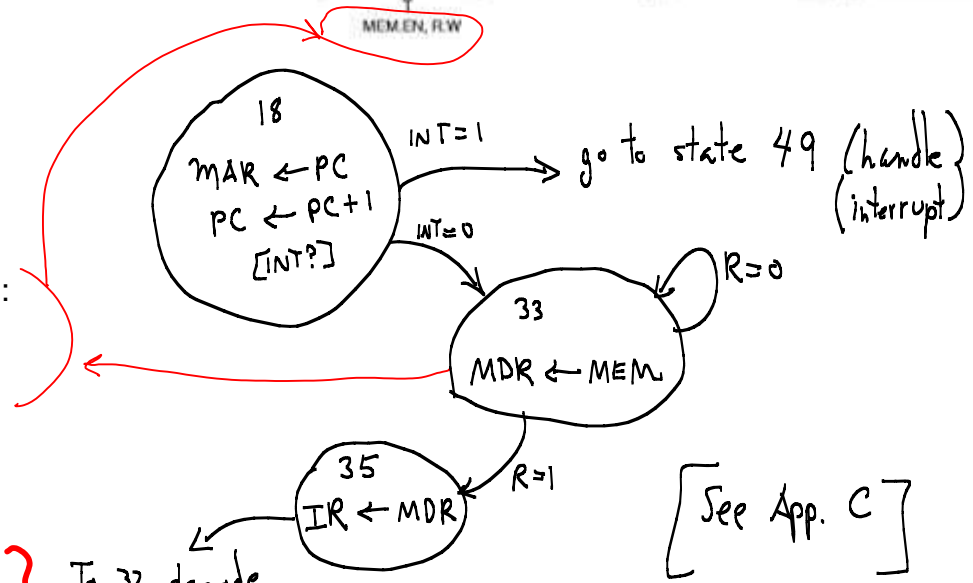


Instruction Formats





NB--State diagram indicates
branch on signal S as, e.g.,
"[S]"
or as labels on arcs, e.g.,
"R=0"



also for state-33:
MIO_EN = 1
R_W = 0

*See Tri-states in the MEMORY-IO BUS

MIO BUS: { databus, address bus, control bus }

[See App. C]

1. See top.Mem-IO bus (address decode, tri-states, control bus).
2. See test.testInstr (initializing memory).

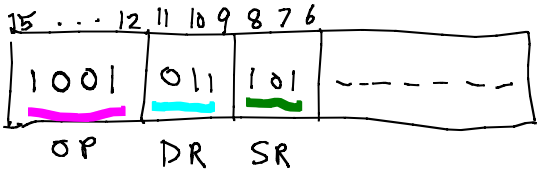
Operate Instructions (operators: ADD, AND, NOT)

NOT

State-9:

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

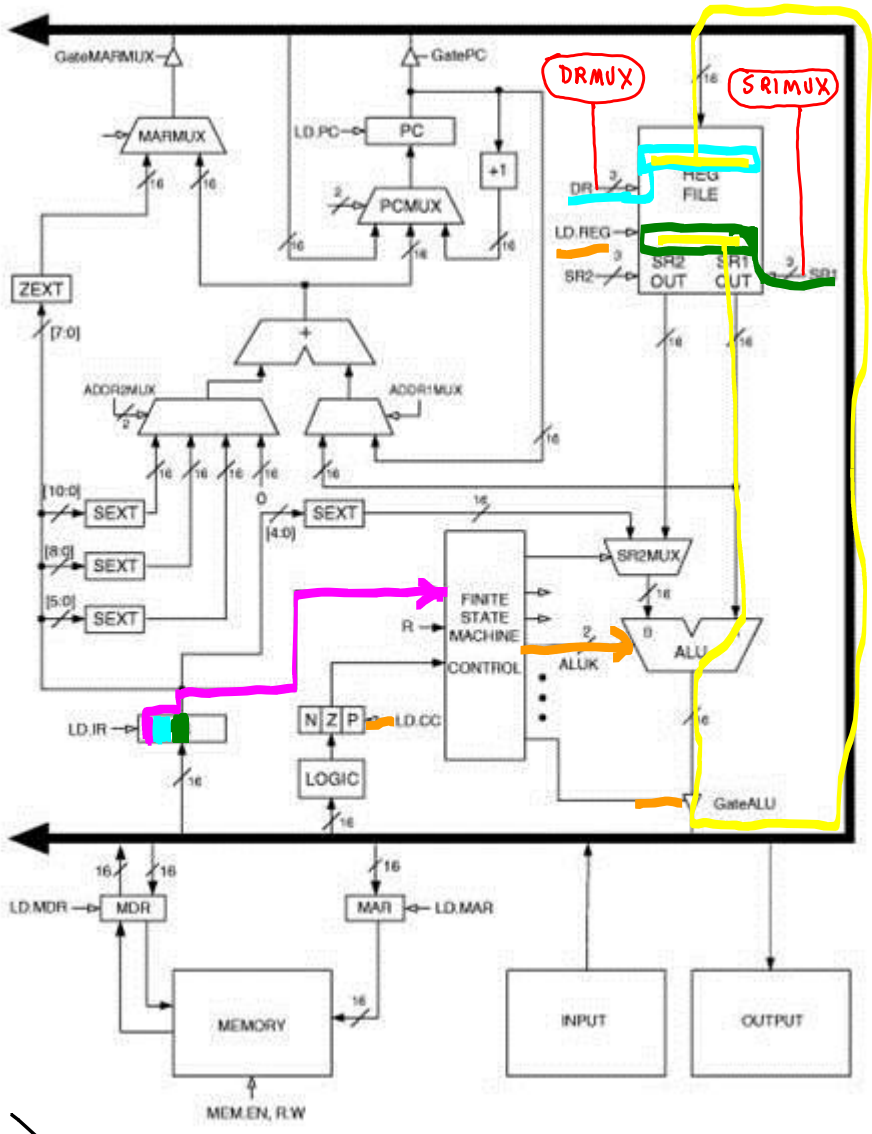
DR <= NOT(SR)
GateALU
LD_REG
LD_CC
ALUK = 10



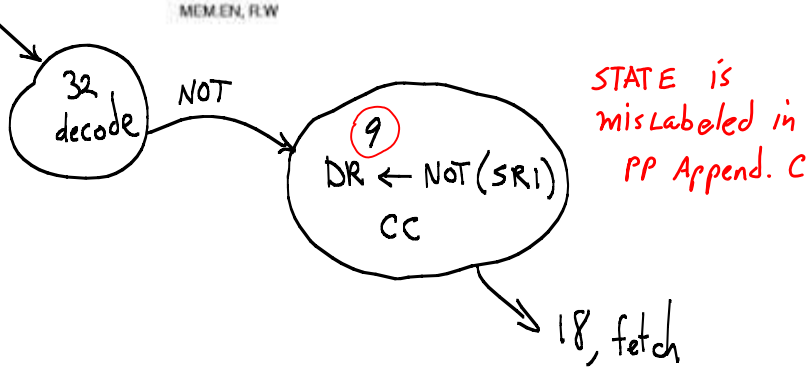
R3 ← NOT(R5)

Register-register addressing

★ MUXed inputs
See p. 574, App. C
Controls: DRMUX[1..0]
SRIMUX[1..0]

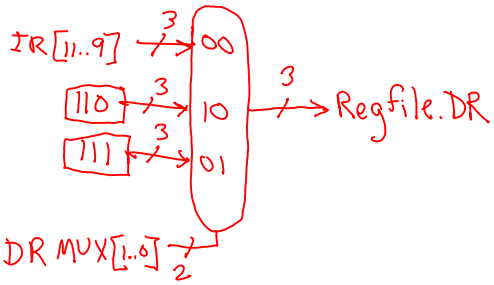


35

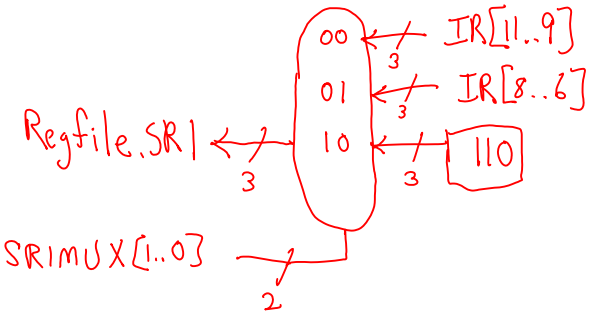


STATE is mislabeled in PP Append. C

DRMUX



SRIMUX



Before:
Regfile[101] = 1100101011110000
After:
Regfile[011] = 0011010100001111
Regfile[101] = 1100101011110000

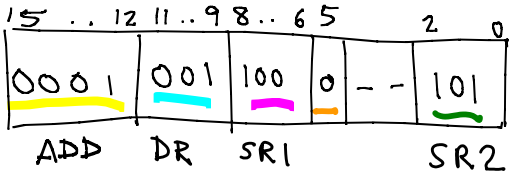
r5 <== NOT(r3)
not r5, r3

ADD (3-register addressing)

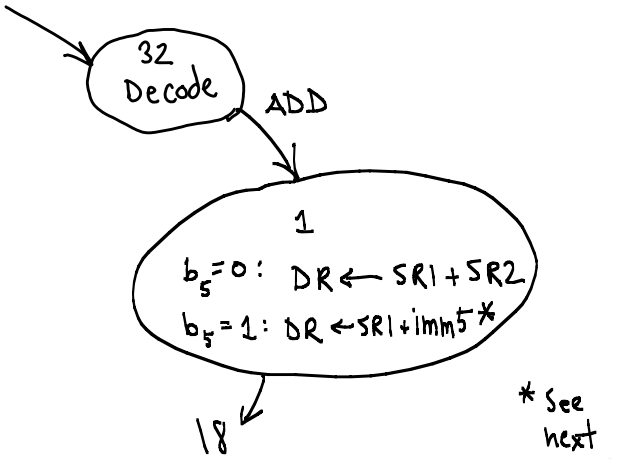
State-1:

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

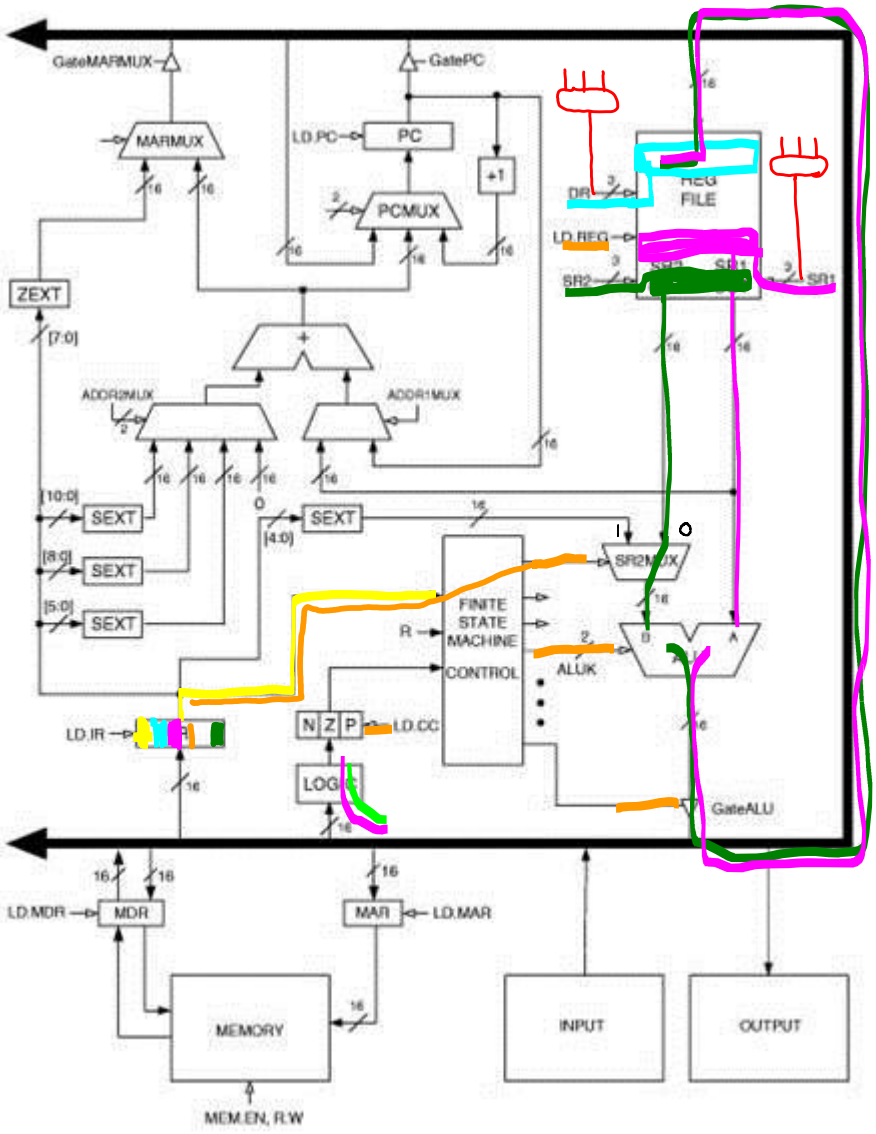
DR <= SR1 + SR2
GateALU
LD_REG
LD_CC
ALUK = 00



R1 ← R4 + R5



* See next page



A - B ? How about A + (-B) using 2s-complement?

1 0 0 1 0 0 1 0 0 1 1 1 1 1 1
NOT R1 (DR) R1 (SR1) 'ignore?

R1 ← NOT (R1), B in R1 → $\bar{B}$

0 0 0 1 0 1 0 0 0 1 0 0 0 1 1
ADD R2 (DR) R1 (SR1) ? (SR2MUX) R3 (SR2)

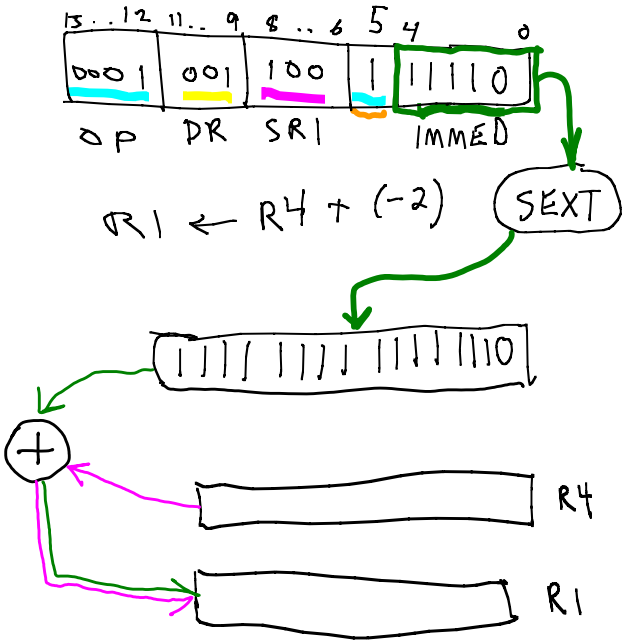
R2 ← R1 + R3, 1 in R3 (how?) -B in R2

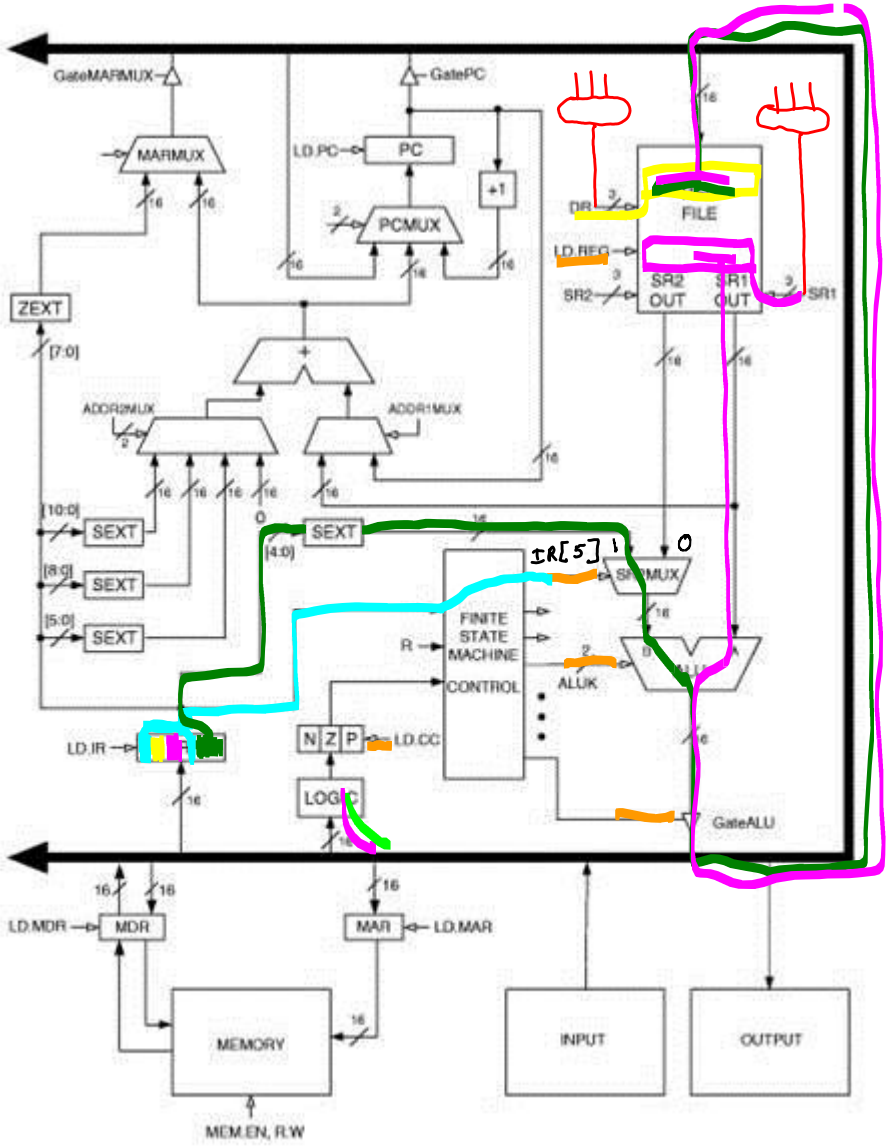
0 0 0 1 0 1 0 0 0 0 0 1 0

R2 ← R0 + R2, A in R0 (A-B) in R2

ADD (2-register/immediate addressing)

State-1:
IR[15..12] -> FSM.in
IR[11..9] -> RegFile.DR
IR[8..6] -> RegFile.SR1
IR[5] -> SR2MUX
IR[4..0] -> SEXT.in
DR <= SR1 + SR2
GateALU
LD_REG
LD_CC
ALUK = 00





A - B ? How about A + (-B) using 2s-complement with immediate constants?

1001 001 001 111111
ADD DR SR1

R1 ← NOT(R1), R1 ← $\overline{B}$

0001 010 001 1 00001
ADD DR SR1 IMM5
SR2MUX

R2 ← R1 + SEXT(00001), R2 ← $(\overline{B}+1)$

0000 0000 0000 0001

0001 010 000 0 010
ADD DR SR1 SR2
SR2MUX

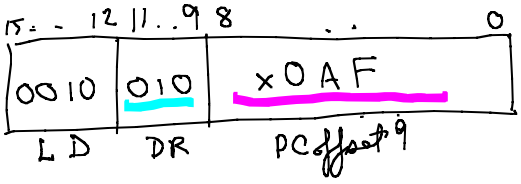
R1 ← R0 + R2, R1 ← (A - B)

(See Appendix A.3)

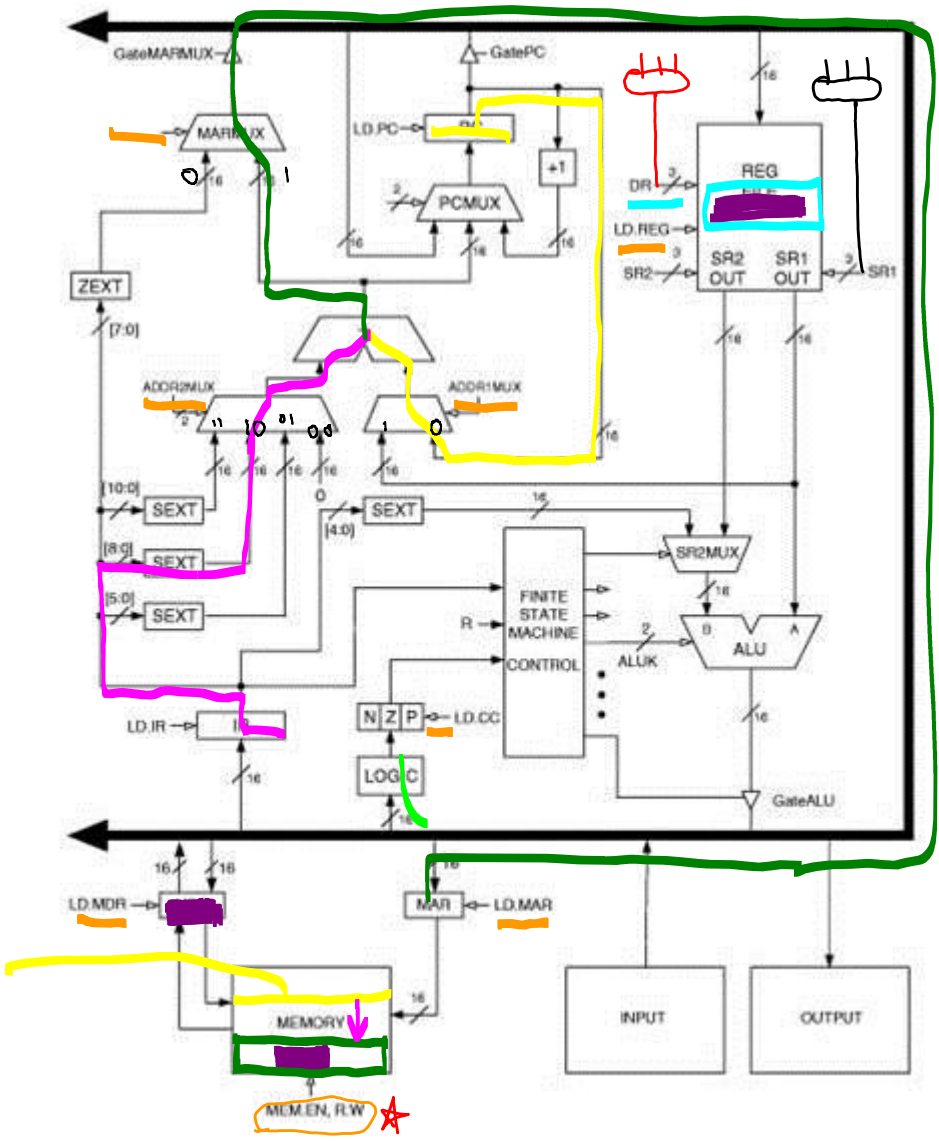
Load/Store (load a register from memory, store register contents in memory)

LD / ST (pc-relative addressing)
LD
State-2:
IR[8..0] -> SEXT
PC -> ADDR1MUX
SEXT -> ADDR2MUX -> MARMUX
MAR <= PC + IR[8..0]
GateMARMUX
LD_MAR

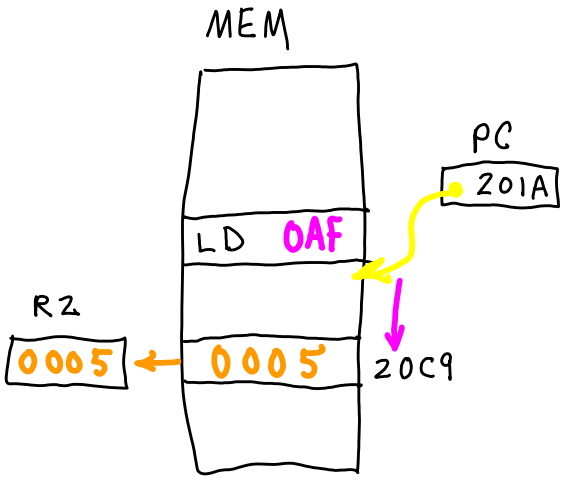
State-25:
MDR <= MEM.out
LD_MDR
MIO_EN
R_W
State-27:
DR <= MDR
GateMDR
LD_REG
LD_CC
DRMUX == ?

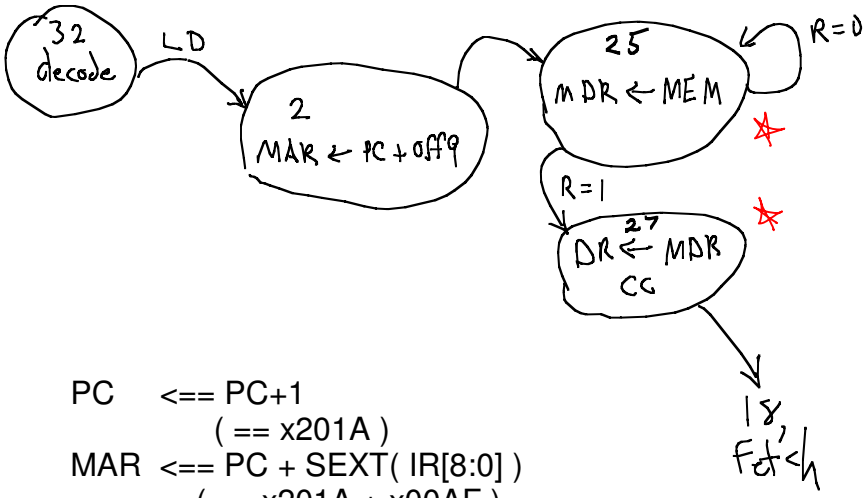


(x0AF is pos.)



fetch
PC ← PC+1





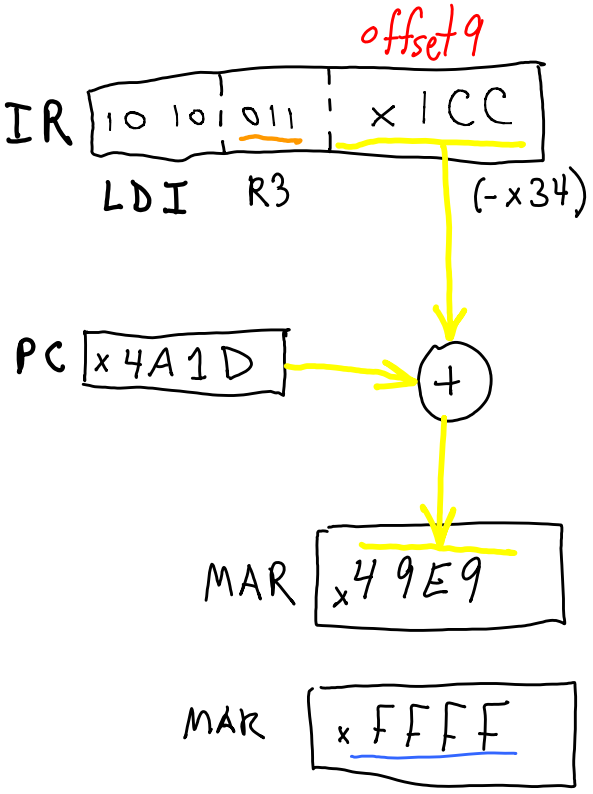
x2019: 0 0 1 0 0 1 0 0 1 0 1 0 1 1 1 1
x201A: LD R2 x0AF

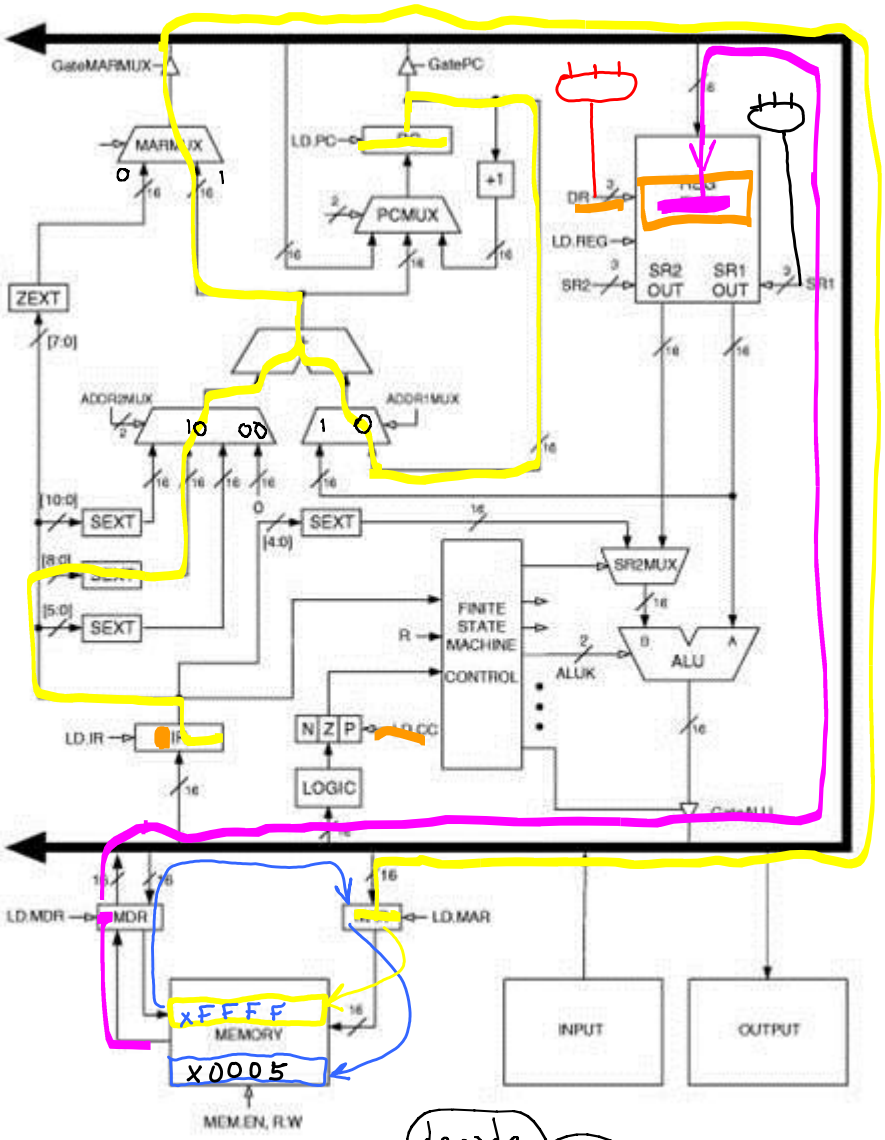
+ x0AF

x20C9: 0000 0000 0000 0101

PC <== PC+1
(== x201A)
MAR <== PC + SEXT(IR[8:0])
(== x201A + x00AF)
(== 0010000000011010 + 00000000101011)
(== x20C9)
MDR <== MEM[x20C9] (== x0005)
R2 <== MDR (== x0005)

LDI / STI (memory indirect addressing)
LDI:
State-10:
MAR <= PC + IR[8..0]
State-24:
MDR <= MEM[PC + PCoffset9]
State-26:
MAR <= MDR
State-25:
MDR <= MEM[MEM[PC + PCoffset9]]
State-27:
DR <= MDR

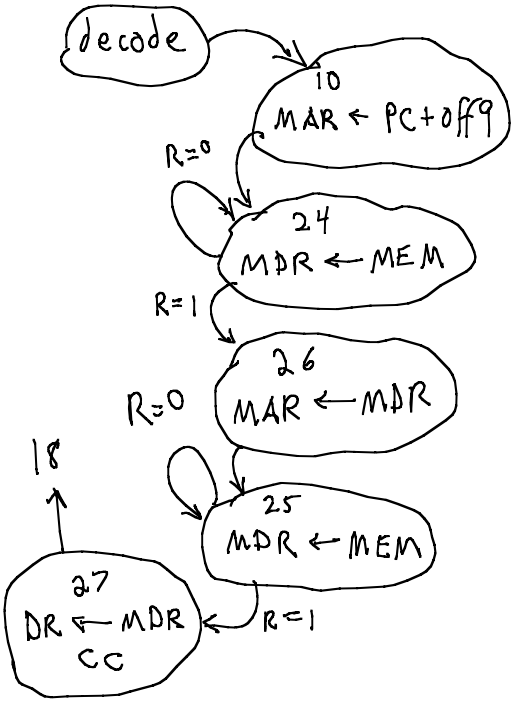


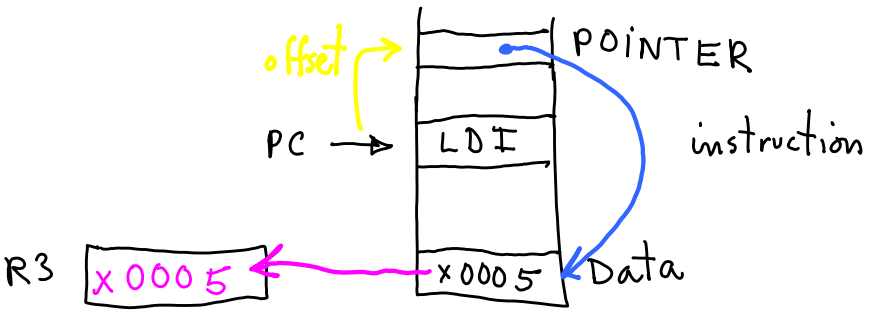


x49E9: 1111 1111 1111 1111 (xFFFF)

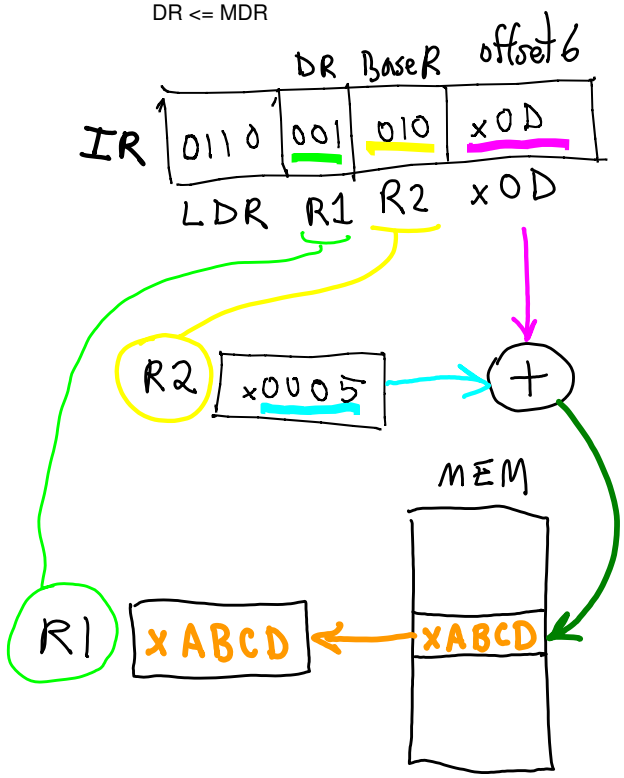
x4A1C: 1010 011 1 1100 1100 (PC <== x4A1D)
LDI R3 offset9
x4A1D:
MAR <== x4A1D + SEXT(x1CC)
= 0100 1010 0001 1101 + 1111 1111 1100 1100
= x4A1D - x0034
= x49E9
MDR <== xFFFF
MAR <== xFFFF
MDR <== x0005
R3 <== MDR

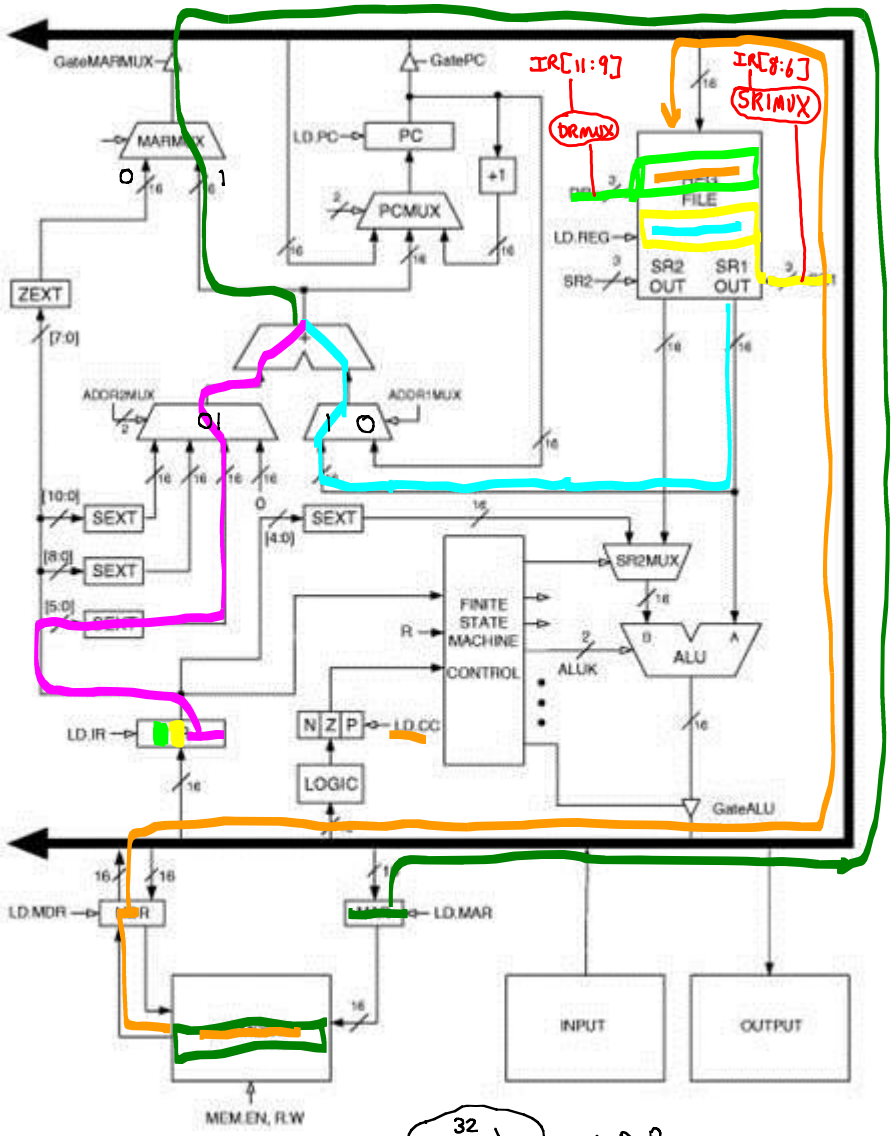
xFFFF: 0000 0000 0000 0101





LDR / STR (register-indirect addressing)
LDR
State-6:
IR[8..6] -> RegFile.SR1
RegFile.SR1.out -> ADDR1MUX
IR[5..0] -> ADDR2MUX
MAR <= BaseR + offset6
State-25:
MDR <= MEM[MDR]
State-27:
DR <= MDR





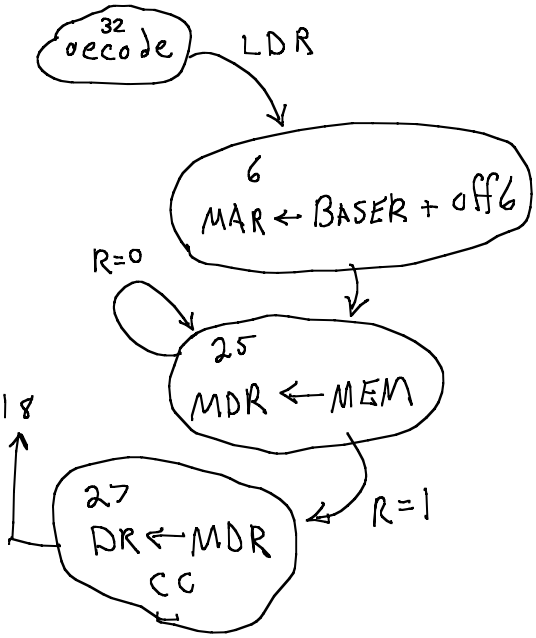
x0005: (BaseR = R2 = x0005)
...
x0012: xABCD
...
x0200: 1010 001 010 001101
LDR DR BaseR off6

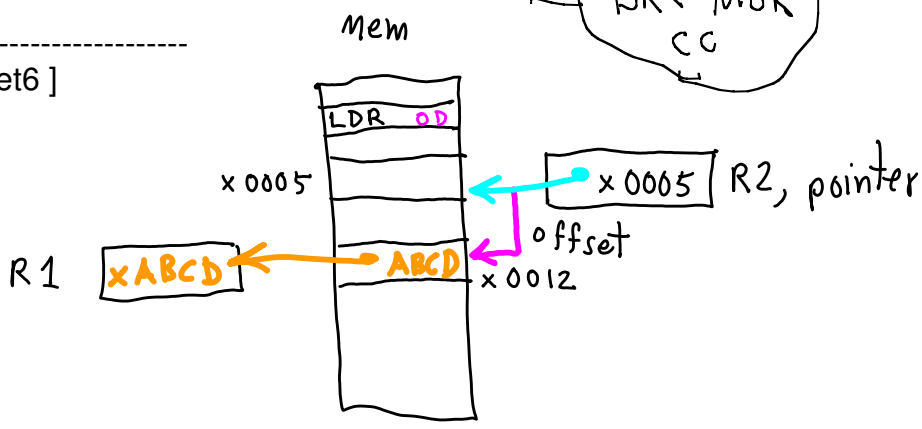
MAR <== R2 + SEXT(x0D) (== x0005 + x000D)
(== x0012)

MDR <== MEM[x0012] (== xABCD)

DR <== MDR (R1 <== xABCD)

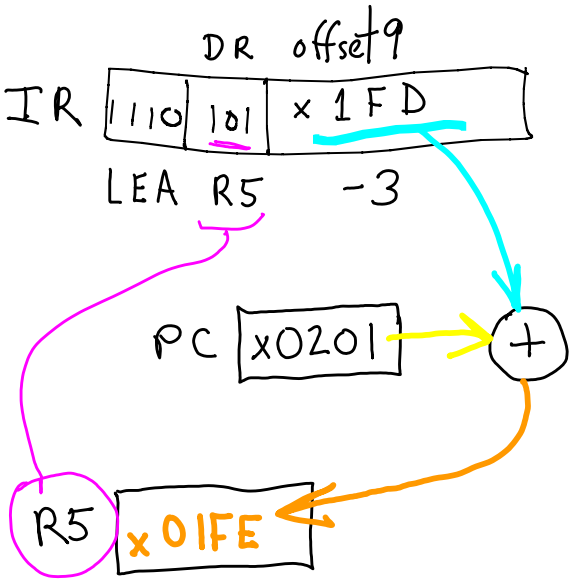
DR <== MEM[BaseR + Offset6]

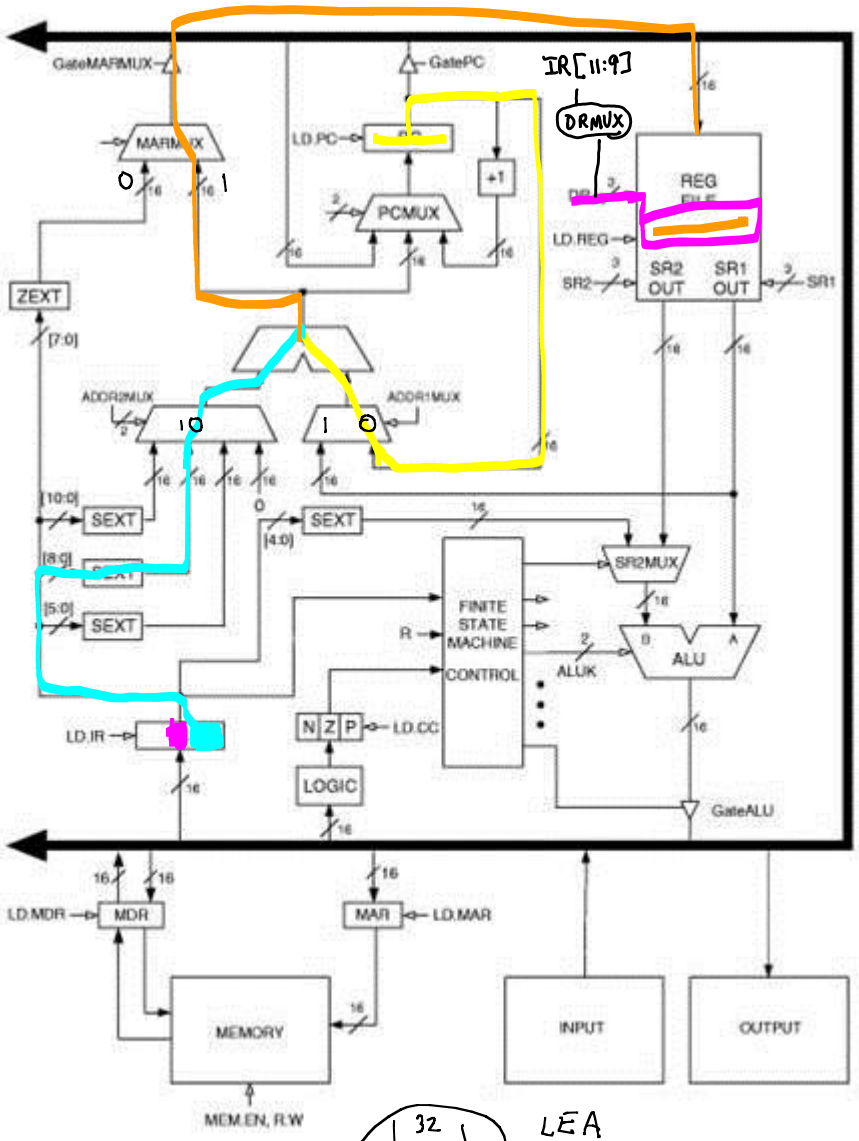


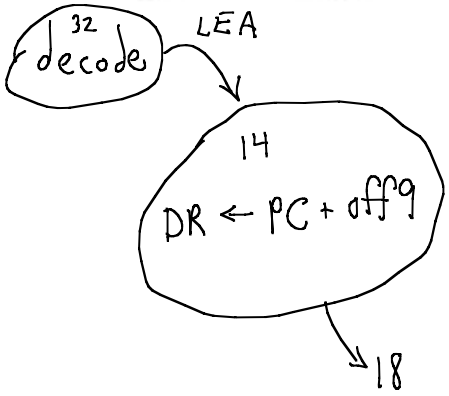


LEA (immediate addressing)

State-14:
PC -> ADDR1MUX
IR[8..0] -> ADDR2MUX
MARMUX -> RegFile.in
DR <= PC + PCoffset9







x01FE:
x01FF:
x0200: 1110 101 111111101 (PC <== x0201)
x0201:
R5 <== PC + SEXT(x1FD)

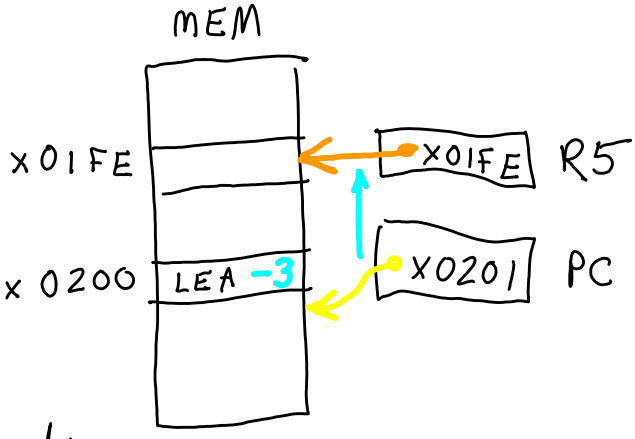
LEA DR off9

1 ~~1111 1101~~

x0201 + xFFFD = x0201 - 3 = ?

x01FE:
x01FF: -1
x0200: -1
x0201: -1

DR <== PC + Offset9 = x01FE



R5 "points to" cell at location x01FE