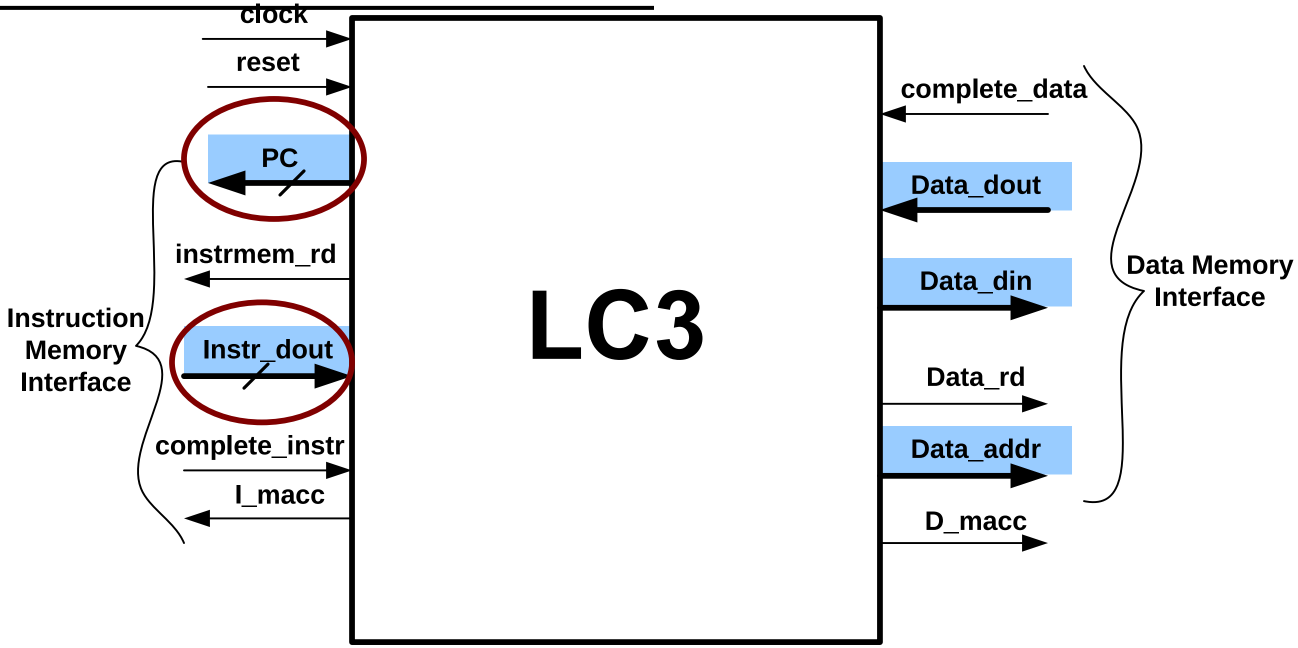
ECE745: PROJECT1: LC3 INSTRUCTION SET ARCHITECTUE

Disclaimer

All the contents in this presentation have been repeated (in large part) with permission from ECE406 Notes for Spring 2007, ECE NCSU (Dr Rhett Davis, Dr Xun Liu).

Top level view of LC3



```
Instruction Register = IR <= IMem_dout
If(DMem_rd), DMem_dout <= DMem[Dmem_addr]
If((IMem_rd), IMem_dout <= IMem[PC]
```

The value read from Instruction Memory provides the relevant information for the LC3 to configure itself in a specific manner. This is the INSTRUCTION and forms the INSTRUCTION REGISTER (IR)

Instructional Example

- Say, R0 to R7 are 8 locations where we can store data
- Each Location can be either written to or read from

(**AND** R0 R0 #0) → (**ADD** R2 R0 #2)→(**ADD** R1 R2 R0)

- PC = **3000** → Instr_dout = IMEM[**3000**] = **5020**
- Source 1 = **R0** ; Source 2 = **Immediate(from IR) #0**; Dest = **R0**
- Operation Performed = **R0 ← R0 & 0 = 0**

- PC = **3001** → Instr_dout = IMEM[**3001**] = **1422**
- Source 1 = **R0** ; Source 2 = **Immediate(from IR) #2**; Dest = **R2**
- Operation Performed = **R2 ← R0 + 2= 2**

- PC = **3002** → Instr_dout = IMEM[**3002**] = **1280**
- Source 1 = **R2** ; Source 2 = **R0**; Dest = **R1**
- Operation Performed = **R1 ← R2 + 0 = 2 + 0 = 2**

LC-3 Instruction Set Architecture

- Memory
  - Each word is 2-bytes wide
  - Separate memory for instructions and data
  - Each memory space has 2^{16} words
- 8 general purpose registers (Register File)
  - Can be a source/destination
- Data type: 2s complement integers
- 15 instructions

  [✓ a) ALU] b) Control [✓ c) Memory]
- Status register:
  - 3-bit **NZP**: **N**egative **Z**ero **P**ositive

    Captures the nature of latest write into register file
- PC: Program Counter: Points to instruction memory *i.e.* instruction = InstrMem[PC]

LC-3 Instruction Set Architecture

- Opcode: unique for each instruction
  - IR[15:12]; IR = instr. register = **IMEM**[PC]
- Addressing mode
  - Immediate (with sign extension) ([DR] ← [SR1] & signextend(Imm))
  - Register (SR1 SR2) ([DR] ← [SR1] + [SR2])
  - Memory:
    - PC relative **DMEM**[PC +1+ signextend(Offset)]
    - Indirect **DMEM**[DMEM[PC+1+signextend(Offset)]]
    - Base register +offset
      DMEM[BaseR + signextend(Offset)]

ALU Operation Instructions

	15 12	11 9	8 6	5	4 3	2 0
ADD	0 0 **0** **1**	DR	SR1	0	0 0	SR2
	0 0 **0** **1**	DR	SR1	1	imm5	
AND	0 1 **0** **1**	DR	SR1	0	0 0	SR2
	0 1 **0** **1**	DR	SR1	1	imm5	
NOT	1 0 **0** **1**	DR	SR1	111111		

ALU Operation Instructions

- Also called "Operate" Instructions
- AND IR[15:12] = **0101**
  - AND DR, SR1, SR2 ;**IR[5]=0**; ([DR]←[SR1]**&**[SR2])
  - AND DR, SR1, Imm5 ;**IR[5]=1**; ([DR]←[SR1]**&** Imm5)
- ADD IR[15:12] = **0001**
  - ADD DR, SR1, SR2 ;**IR[5]=0**; ([DR]←[SR1]**+**[SR2])
  - ADD DR, SR1, Imm5 ;**IR[5]=1**; ([DR]←[SR1]**+** Imm5)
- NOT IR[15:12] = **1001**
  - NOT DR, SR1 ([DR] ← **~**[SR1])
- Imm5 is going to be sign extended to 16 bits for all computation- maintain sign for 2's complement manipulation.

ALU Example

(IR) = **16’h12BC,** assume R2 = **16’h0030 = 48** (decimal)

15 12	11 9	8 6	5	4 3	2 0
0 0 **0** 1	0 0 1	0 1 0	1	1 1	1 0 0
	DR	**SR1**			**SR2**

- **IR[15:12] = 0001** => operation is an ADD;
- **DR = IR[11:9] = 001** => we are writing to i.e. destination Register = R1;
- **SR1 = IR[8:6] = 010** => We are reading from i.e. source register 1 = R2;
- since **IR[5] = 1** => Immediate mode, **IR[4:0] = imm5 = 28 = -4**
- Resulting operation (in hex):
  - **R1← R2 + sxt(-4)**
  - **R1 ← 16’h30 + 16’hFFFC = 002C**(ignoring carry out) **= 44**

Memory Operations

	15 12	11 9	8 6	5 4 3 2 0
LD	0 0 **1 0**	DR	PCoffset9	
LDR	0 1 **1 0**	DR	BaseR	Offset6
LDI	1 0 **1 0**	DR	PCoffset9	
LEA	1 1 **1 0**	DR	PCoffset9	
ST	0 0 **1 1**	SR	PCoffset9	
STR	0 1 **1 1**	SR	BaseR	Offset6
STI	1 0 **1 1**	SR	PCoffset9	

Memory Operations

- Load Effective Address
  - LEA DR, `Offset` `([DR] ←` $\mathbf{PC_{mem}+1}$`+Offset)`

    `Offset = sxt(PCOffset9)`
- Load/Store (PC relative addressing mode)
  - LD DR, `Offset` `([DR] ←` **MEM**$[PC_{mem}+1+Offset]$`)`
  - ST SR, `Offset` `(`**MEM**$[PC_{mem}+1+ Offset]$ `← [SR])`

    `Offset = sxt(PCOffset9)`
- Load/Store Register (Base+Offset addressing mode)
  - LDR DR, BaseR, `Offset`

    `([DR] ←` **MEM**`[BaseR+Offset])`
  - STR SR, BaseR, `Offset`

    `(`**MEM**`[BaseR+Offset] ← [SR])`

  `Offset = sxt(PCOffset6)`

Memory Operations

- Load/Store Indirect (indirect addressing mode)
  - LDI DR, `Offset`

    $(\mathtt{DR} \leftarrow \mathbf{MEM}[\boxed{\mathbf{MEM}[\mathtt{PC}_{\mathtt{mem}}\mathtt{+1+Offset}]}])$
  - STI SR, `Offset`

    $(\mathbf{MEM}[\mathbf{MEM}[\mathtt{PC}_{\mathtt{mem}}\mathtt{+1+Offset}]] \leftarrow \mathtt{SR})$

`Offset = sxt(PCOffset9)`

LEA Example

- (IR) = **16’hEDFE**, **PC_{mem}** = **16’h310C**,

15 12	11 9	8 6 5 4 3 2 0
1 1 **1 0**	1 1 0	1 1 1 1 1 1 1 1 0

- **IR[15:12] = 1110** => operation is an LEA;
- **DR = IR[11:9] = 110** => we are writing to i.e. Destination Register = R6;
- **Mem_Addr = PC_{mem}** + 1 + sign-extended(*PCoffset9*)
- **Mem_Addr = 310C + 1 + sxt(-2)**
  = 310C + 1 + sxt(1FE)
  = 310C + 1 + FFFE = 310B
- Resulting operation (in hex):
  - **R6 ← Mem_Addr**
  - **R6 ← 16’h310B**

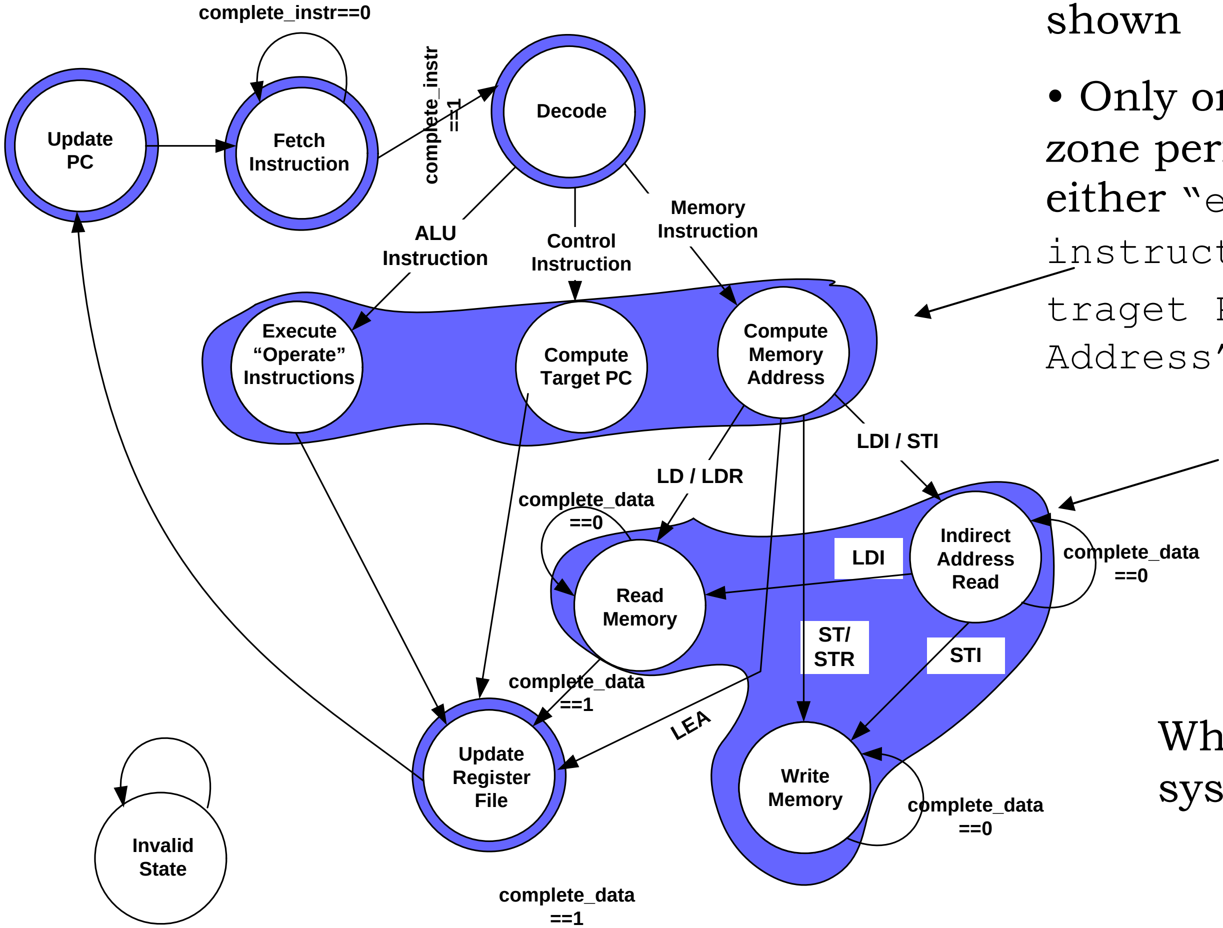
Project1 Specifics

- Subset of instructions
  - ALU (ADD, NOT, AND) ; LEA
  - All instructions take 5 clock cycles ignoring `complete_instr`
  - Some inputs and outputs invalid: Will become clear in next project(s)
- LC3 is unpipelined
  - Each instruction goes through the 5 cycles

`Fetch → Decode → Execute → Writeback → UpdatePC`

  - No typical pipeline issues
    - NO control and data dependence
    - NO stalling requirements
    - FOR NOW !!

LC-3 Components (Base States)



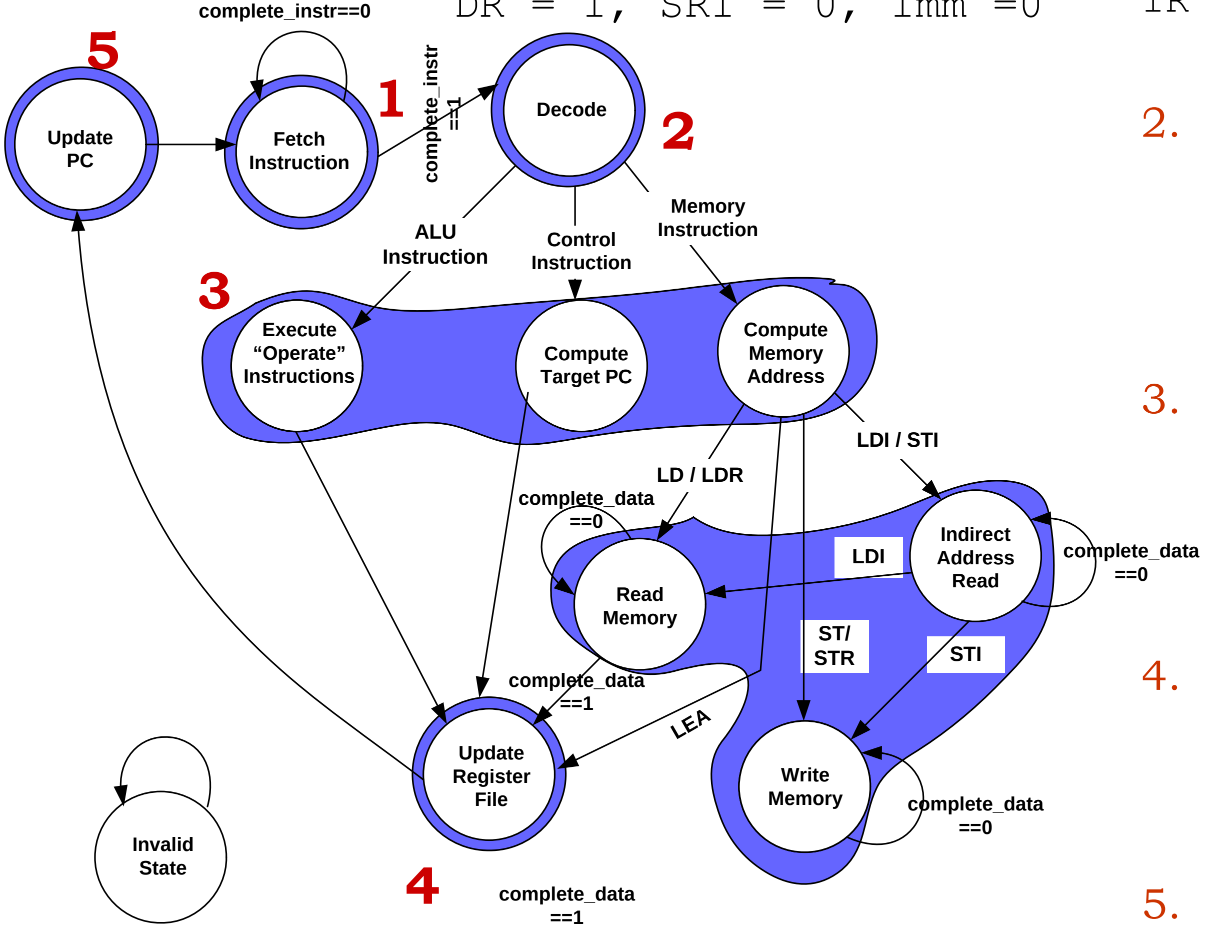
- 5 base pipeline "zones" + memory shown
- Only one of the operations in a zone performed in a clock cycle eg. either "execute operate instruction" OR "compute traget PC" OR "Compute Memory Address"

You will never come here for this project

What would change if the system was pipelined?

ADD/NOT/AND Base States

```
Example:
IR = 5220 (AND R1, R0,#0)
DR = 1, SR1 = 0, Imm =0
```



1. Fetch Unit loads instruction from memory

   `IR = IMem[PC] = 5220`

2. Decode Unit determines the operands using `IR`

   `E_control, W_Control created`

3. Execute Unit applies operands to ALU using `E_Control`

   `aluout = R0 & 0`

4. Result stored in Register File

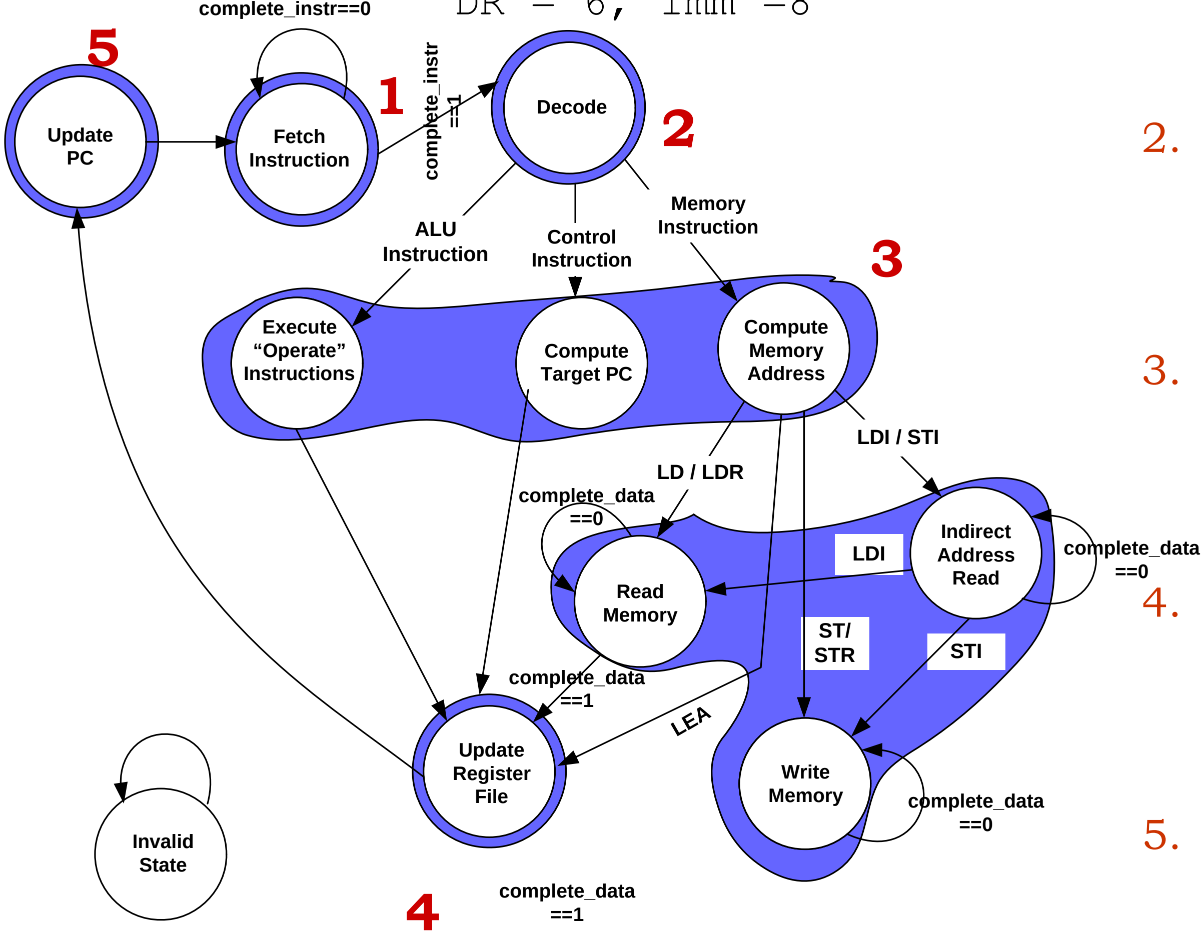
   `R1 = aluout`

5. PC incremented

   `PC = PC + 1`

LEA Base States

```
Example:
IR = EC08 (LEA R6,#8)
DR = 6,  Imm =8
```



1. Fetch Unit loads instruction from memory

`IR = IMem[PCmem] = EC08`

2. Decode Unit determines the operands

`E_control, W_Control etc.`

3. Compute Memory Address with Offset

`pcout = PCmem+1+8`

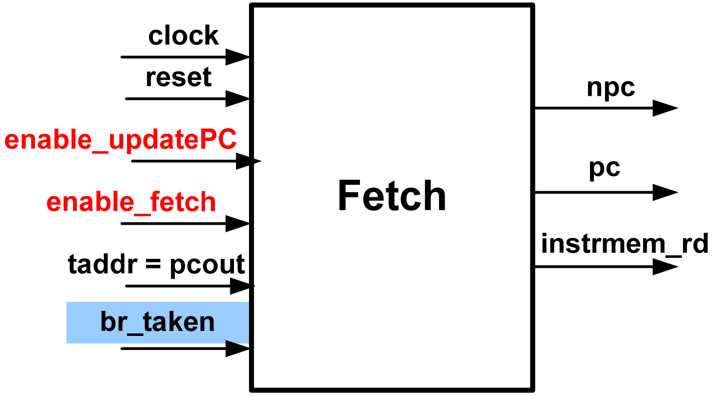
4. Write Memory Address to Reg File

`R6 = pcout`

5. PC incremented

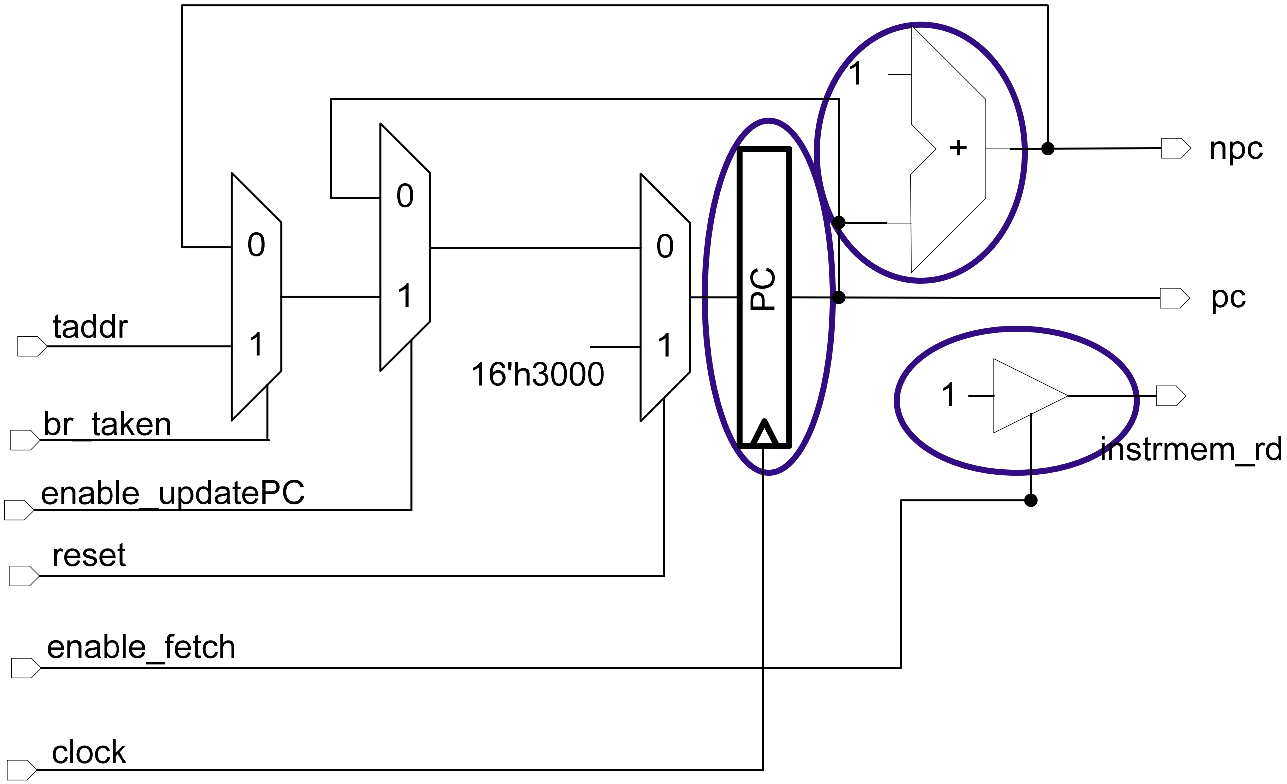
`PC = PC + 1`

LC3 Internals FETCH

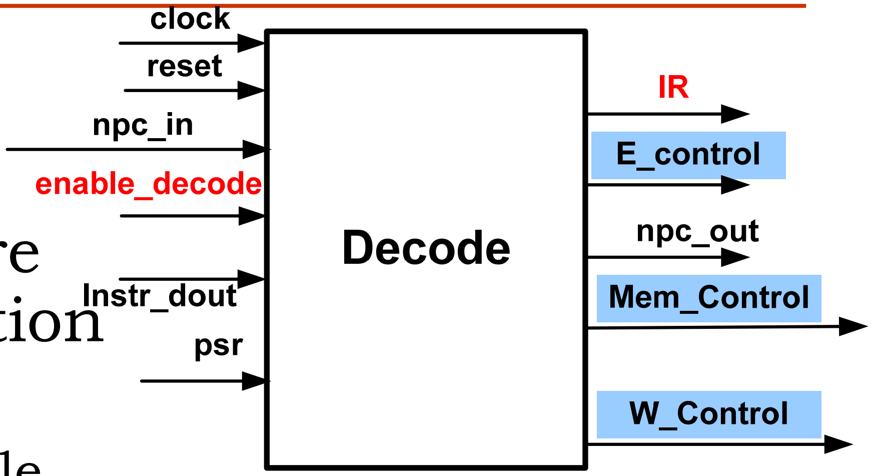


- Creates the correct PC to read correct Instruction from IMEM
- Creates Imem read enable instrmem_rd
- On reset: pc =3000; npc =3001; instrmem_rd = 1
- Signals of Importance
  - enable_fetch and enable_updatePC: Master Control
  - PC = Program Counter of instruction being fetched
  - npc = pc + 1 asynchronously
  - instrmem_rd = enable for Instruction Memory Read
- Ignore
  - br_taken = if 1 PC = taddr else PC = PC+1
  - taddr = new PC if branch is taken
- if enable_fetch=1 instrmem_rd=1 **asynchronously** else instrmem_rd=Z (HIGH IMPEDANCE)
- PC, updated **synchronously** only when enable_updatePC=1 => npc updated **asynchronously**

LC3 Internals FETCH



LC3 Internals DECODE



- Creates relevant control signals using Instr_dout
- Controls uniquely configure datapath for given instruction
- Signals of Importance
  - enable_decode: master enable
  - E_Control = Execute Block Control
  - W_Control = Writeback Block Control
  - IR = Instruction Register: Reflects contents of Instr_dout
  - npc_out: reflects contents of npc_in
- Ignore
  - Mem_Control = Control signals for memory related operations
- All outputs created **synchronously** when enable_decode=1
- On reset, all outputs go to 0 synchronously

LC3 Internals DECODE

- W_Control
  - 2 bits
  - Control for Writeback
  - Function of IR[15:12]
  - Enables choice between aluout, memout, pcout for writes to register file
  - We shall focus only on ALU and LEA instructions W_control either 0 (ALU) or 2 (LEA)

ALU ([DR]←[SR1] **aluop** [SR2])
ALU ([DR]←[SR1] **aluop sxt**(Imm))
LEA ([DR] ← PC+1+**sxt**(PCOffset9))

Operation	IR[5]	**W_Control**
ADD	0	**0(aluout)**
	1	**0(aluout)**
AND	0	**0(aluout)**
	1	**0(aluout)**
NOT		**0(aluout)**
BR		0
JMP		0
LD		**1(memout)**
LDR		**1(memout)**
LDI		**1(memout)**
LEA		**2(pcout)**
ST		0
STR		0
STI		0

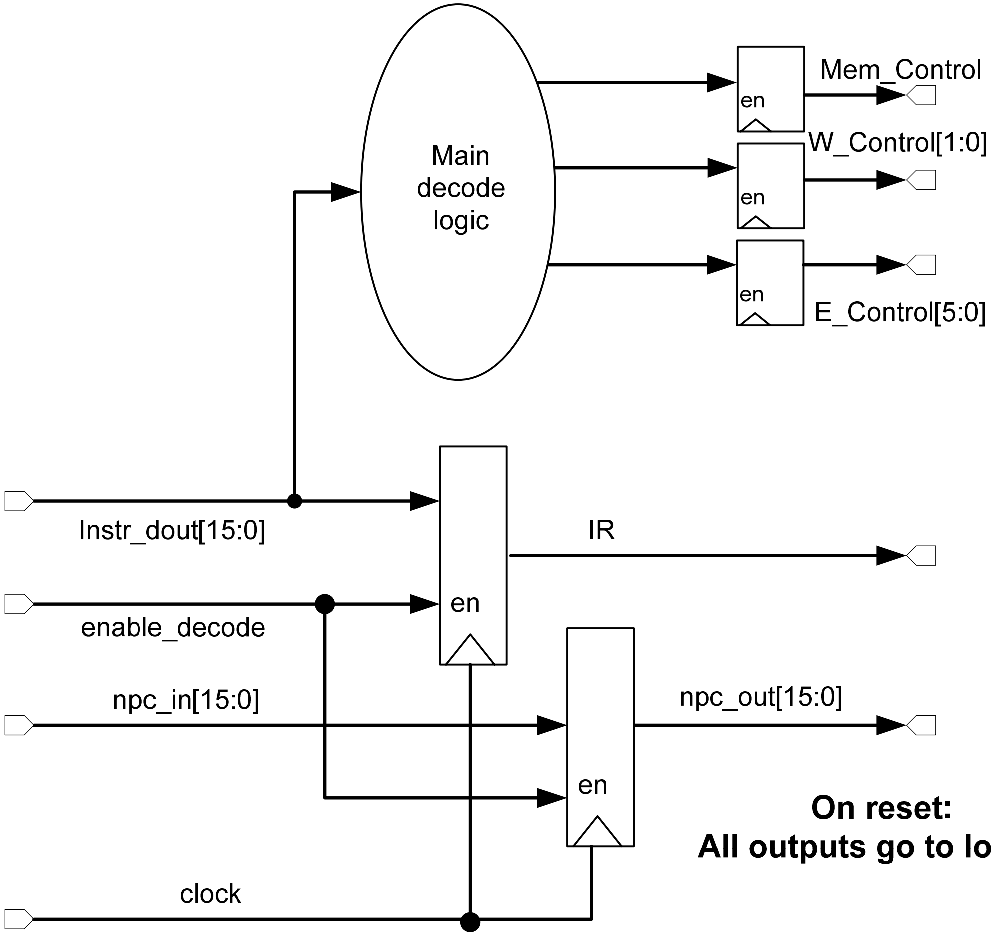
LC3 Internals DECODE

ALU ([DR]←[SR1] **aluop** [SR2]) **ALU** ([DR]←[SR1] **aluop sxt**(Imm))

LEA ([DR] ← PC+1+**sxt**(PCOffset9))

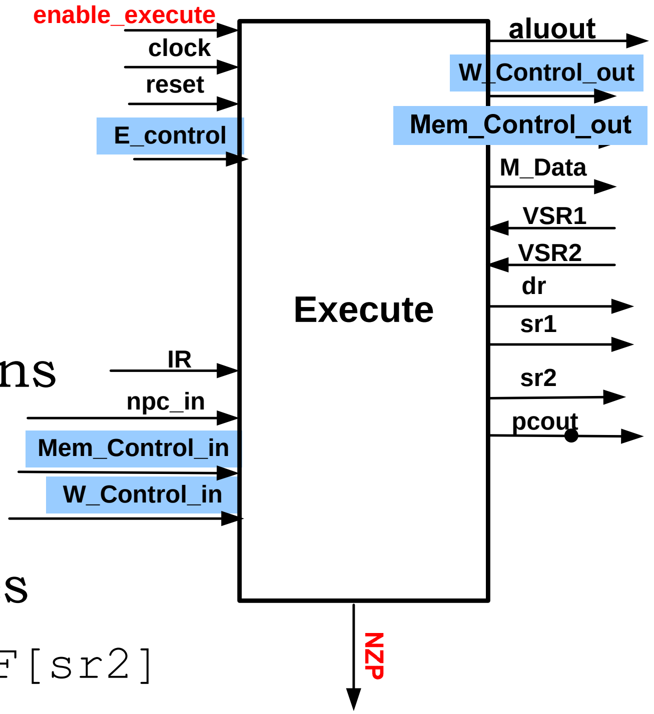
		E_Control			
IR[15:12]	IR[5]	**alu_control [2]**	**pcselect1 [2]**	**pcselect2 [1]**	**op2select [1]**
ADD	0	**0**	-	-	VSR2 **(1)**
	1	**0**	-	-	imm5 **(0)**
AND	0	**1**	-	-	VSR2 **(1)**
	1	**1**	-	-	imm5 **(0)**
NOT		**2**	-	-	-
BR		-	offset9 **(1)**	npc **(1)**	-
JMP		-	0 **(3)**	VSR1 **(0)**	-
LD		-	offset9 **(1)**	npc **(1)**	-
LDR		-	offset6 **(2)**	VSR1 **(0)**	-
LDI		-	offset9 **(1)**	npc **(1)**	-
LEA		-	offset9 **(1)**	npc **(1)**	-
ST		-	offset9 **(1)**	npc **(1)**	-
STR		-	offset6 **(2)**	VSR1 **(0)**	-
STI		-	offset9 **(1)**	npc **(1)**	-

LC3 Internals DECODE



On reset:
All outputs go to logic 0

LC3 Internals EXECUTE



- Closely coupled with the Writeback Block which provides all the relevant data values for register related operations
- Signals of importance
  - sr1 & sr2 = source register addresses
  - VSR1 & VSR2 = values of RF[sr1] & RF[sr2] created **asynchronously** in Writeback
  - aluout = result of alu operation (ADD, NOT, AND)
  - pcout = result of pc related operation (BR,JMP,LEA)
  - dr = destination register address
  - W_control_out: reflects synchronously W_control_in
- Ignore
  - M_Data = RF value to be written to memory
  - Mem_control_in, NZP, Mem_control_out

LC3 Internals EXECUTE

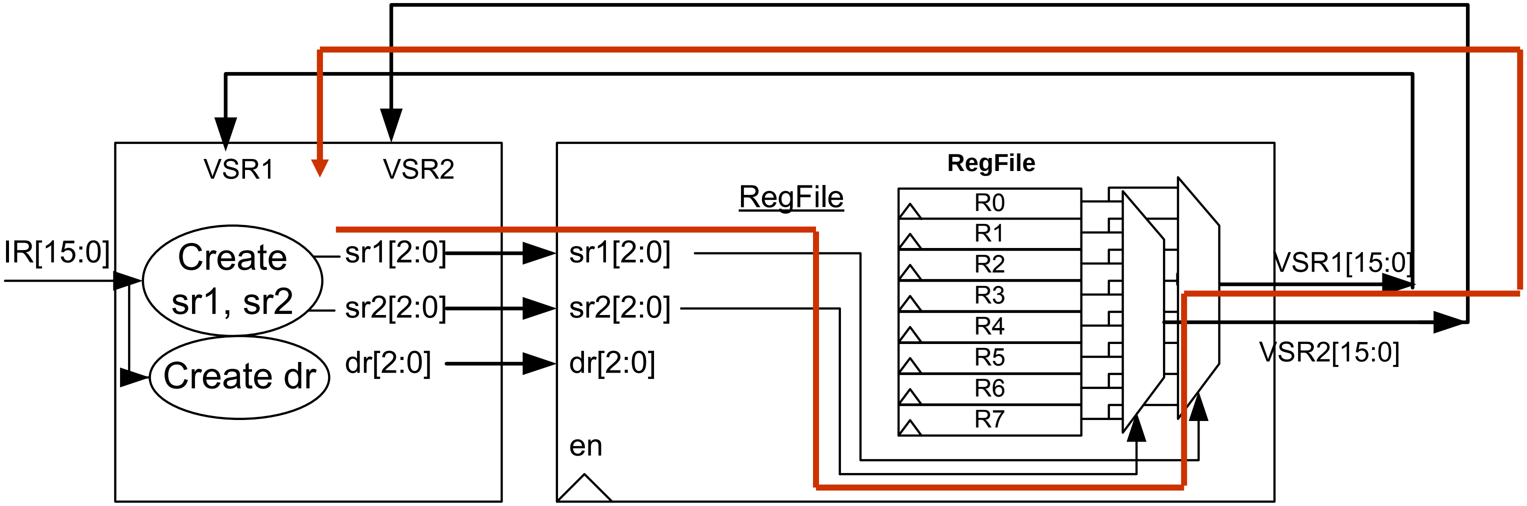
- DR created Synchronously
- SR1/SR2 created asynchronously

ALU ([DR]←[SR1] **aluop** [SR2])

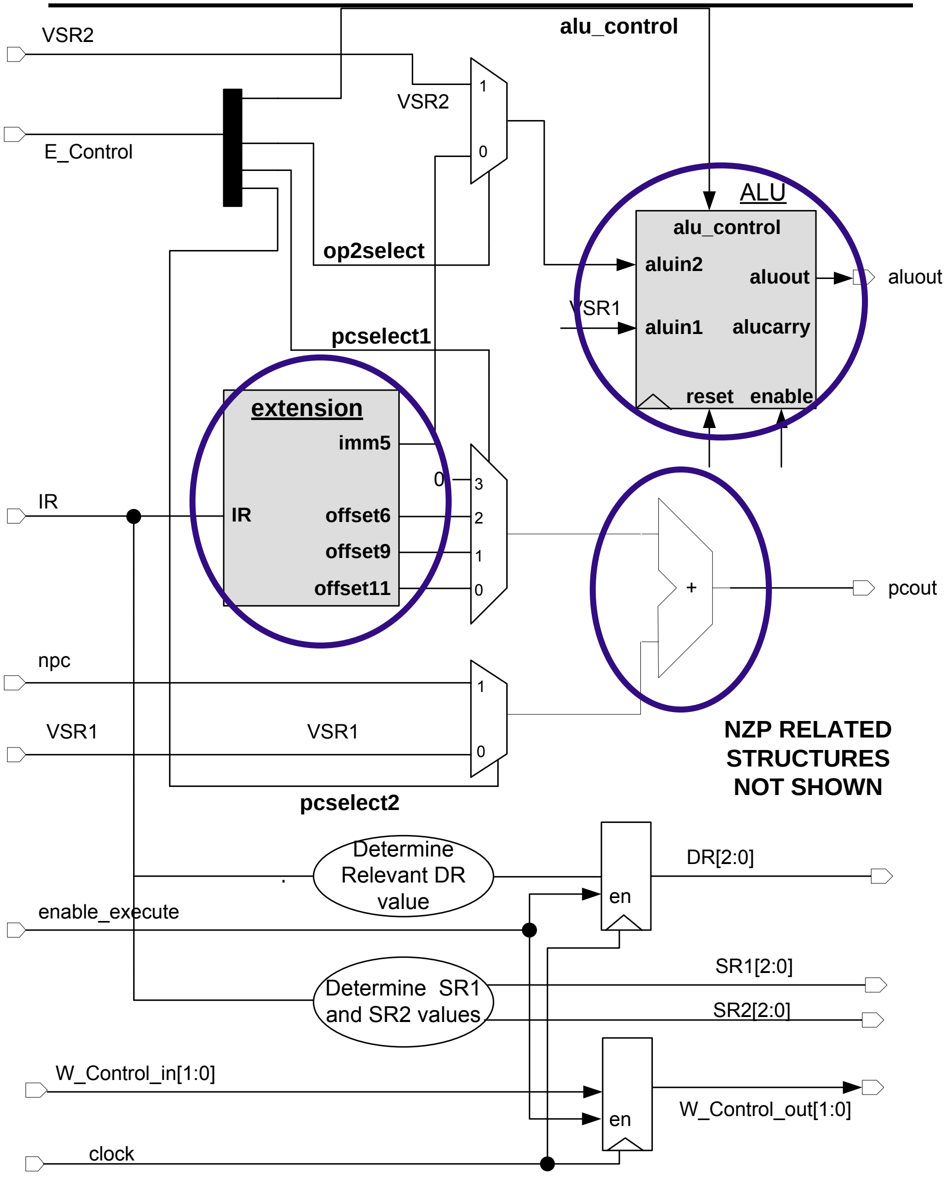
ALU ([DR]←[SR1] **aluop sxt**(Imm5))

LEA ([DR] ← PC+1+**sxt**(PCOffset9))

DR: IR[11:9] (synchronous)
SR1: IR[8:6] (asynchronous)
SR2: IR[2:0] (asynchronous)



LC3 Internals EXECUTE



ALU ([DR]←[SR1] **aluop** [SR2])
ALU ([DR]←[SR1] **aluop sxt**(Imm))
LEA ([DR] ← PC+1+**sxt**(PCOffset9))

imm5 = {11{IR[4], IR[4:0]}

offset6 = {10{IR[5], IR[5:0]}

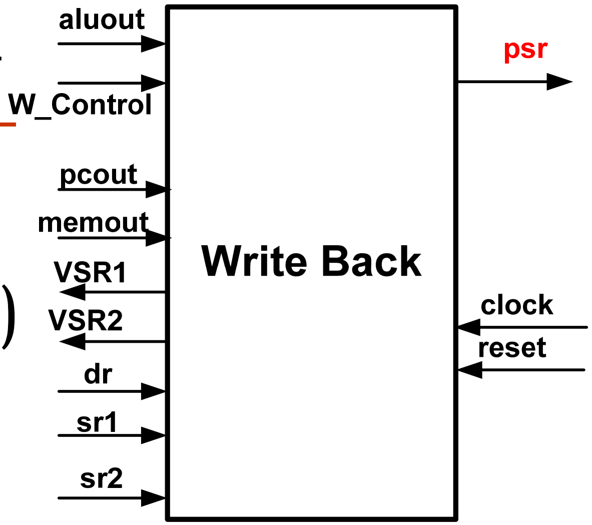
offset9 = {7{IR[8], IR[8:0]}

offset11 = {5{IR[10], IR[10:0]}

On reset,

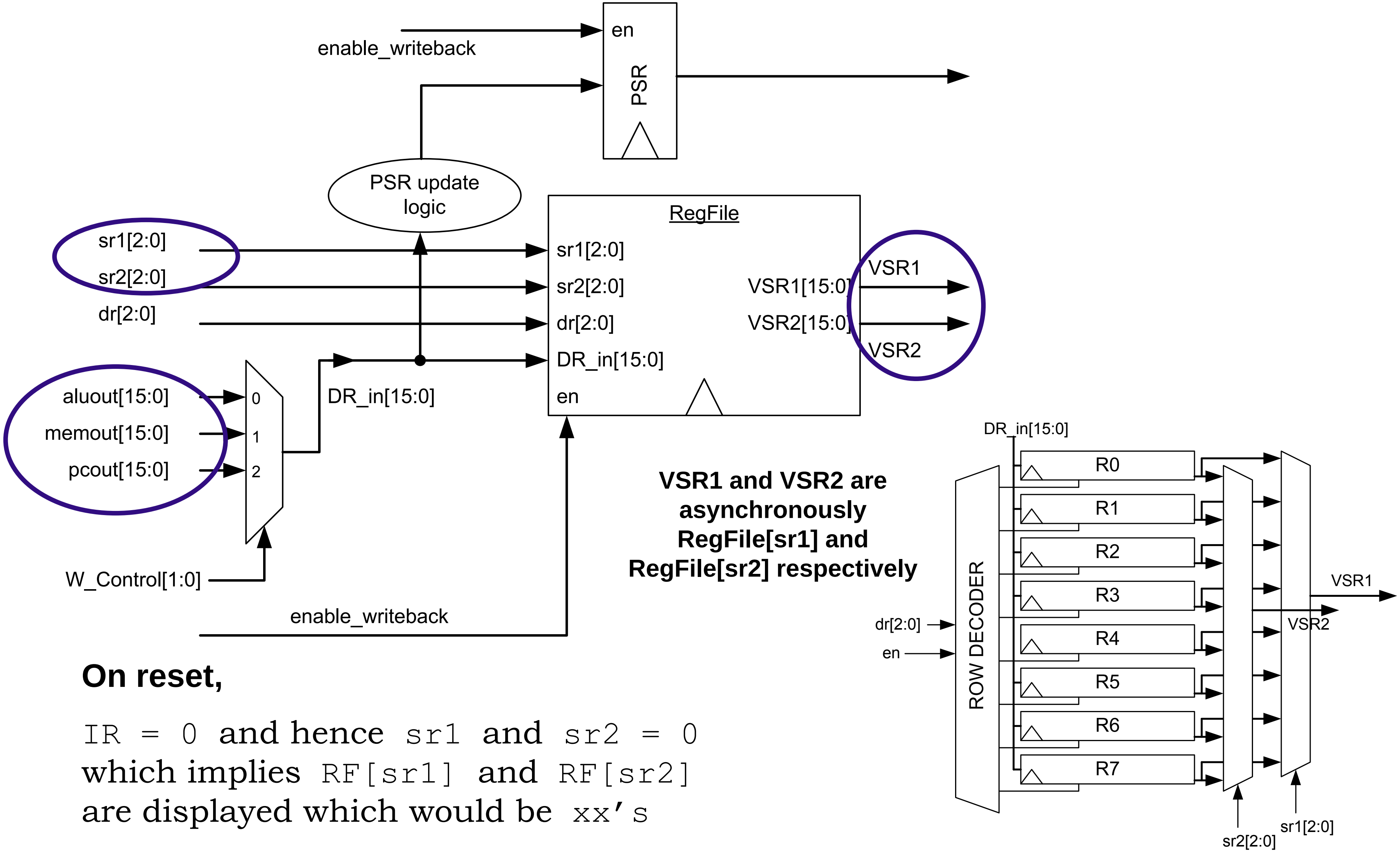
aluout, pcout, W_control_out, dr
go to 0

LC3 Internals WRITEBACK



- Contains register file (RF/RegFile)
  - Addressed using sr1, sr2 and dr
  - Writes either aluout, pcout OR memout based on W_control_in value
  - Synchronous Writes to RF with dr
    RegFile[dr] = DR_in
  - Asynchronous Reads from RF using sr1&sr2
- Important Signals:
  - enable_writeback: master enable
  - aluout, pcout, memout:
  - W_Control
  - sr1, sr2, dr, VSR1, VSR2

LC3 Internals WRITEBACK

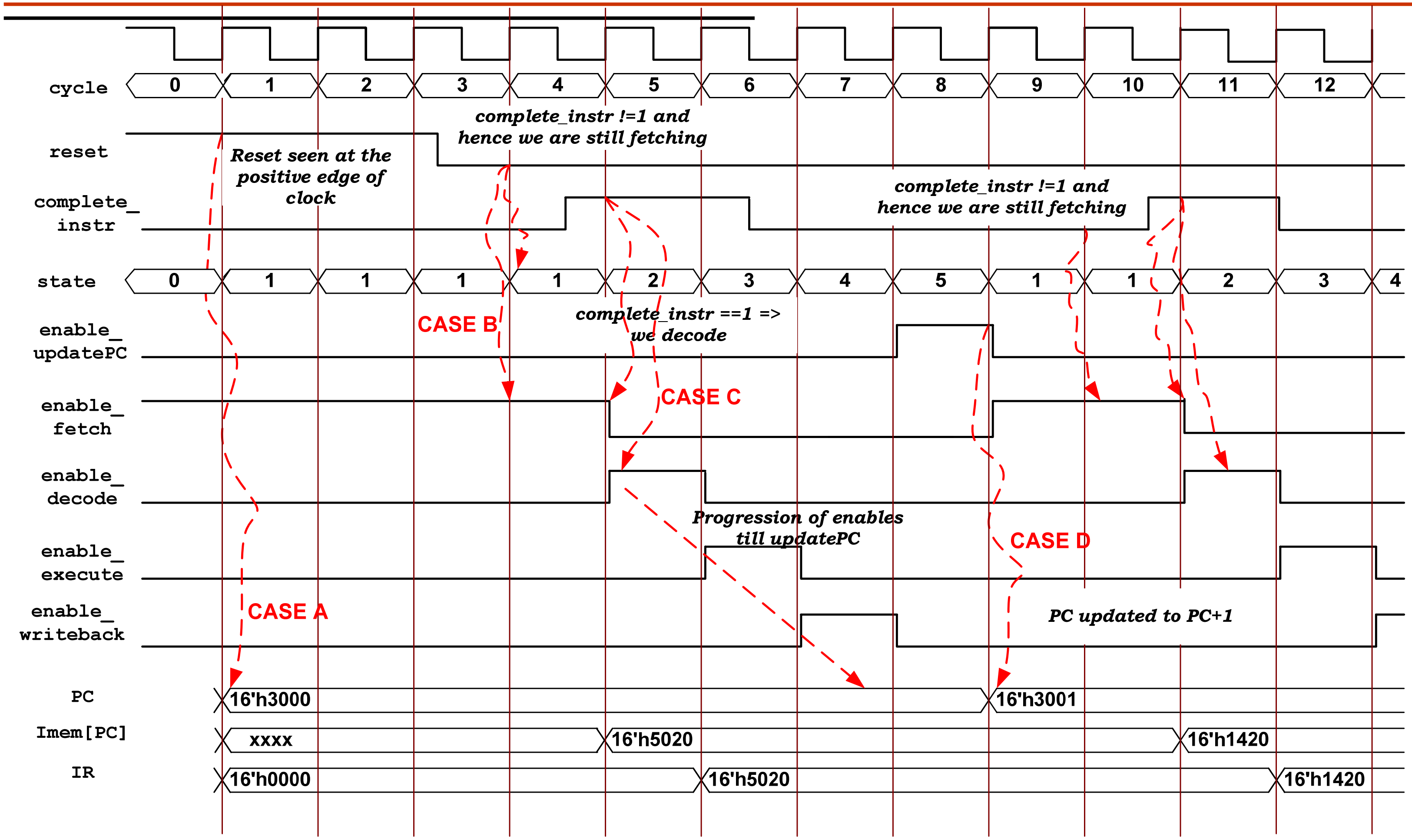


VSR1 and VSR2 are asynchronously RegFile[sr1] and RegFile[sr2] respectively

On reset,

`IR = 0` and hence `sr1` and `sr2 = 0` which implies `RF[sr1]` and `RF[sr2]` are displayed which would be `xx's`

LC3 Internals CONTROL



@3000: 5020 (**AND** R0 R0 #0); **@3001: 1420** (**ADD** R2 R0 #0)

- Sensitvity to `complete_instr` signal only:
  transition from `state`: 1→2 only when `complete_instr == 1` at posedge of clock
- `PC` = Program Counter ; `IR` = Instruction Register ; `IMem` = Instruction Memory ;

Detailed Timing Example

```
@3000: 5020 (AND R0 R0 #0);    @3001: 1422 (ADD R2 R0 #2)
@3002: 1280 (ADD R1 R2 R0);    @3003: EDFE (LEA R6 #-2)

FETCH (after reset has gone to 0)
in: enable_fetch = 1 ; pc = 3000 ; npc = 3001 ;
out: instrmem_rd = 1 ;

DECODER
in: Instr_dout = IMem[3000] = 5020 ;  npc_in  = 3001 ;
    enable_decoder = 1 ;
out: npc_out = 3001; IR = 5020 ; W_Control = 0 (aluout) ;
     alu_control = 1; pcselect1 = 0; pcselect2 = 0; op2select = 0 (imm5)
     => E_Control = [6'b01_0000]

EXECUTE
in: E_Control = [6'b01_0000]; W_Control_in = 0 ; npc_in = 3001 ;
    IR = 5020 ; enable_execute = 1 ;
    VSR1 = R0 = xxx ; VSR2 = R0 = xxx ;
out: sr1 = 0; sr2 = 0 ;
    dr = 0 ; W_Control_out = 0 ; aluout = VSR1 & sxt(Imm5) = R0 & 0  = 0

WRITEBACK
in: dr = 0 ; W_Control = 0 ; aluout = 0 ; enable_writeback = 1 ;
out: RegFile[dr] = aluout = 0 => R0 = 0

UPDATEPC
in: enable_updatePC = 1 ; out: pc = 3001 ; npc = 3002
```

Detailed Timing Example

@3000: 5020 (**AND** R0 R0 #0); **@3001: 1422** (**ADD** R2 R0 #2)

@3002: 1280 (**ADD** R1 R2 R0); **@3003: EDFE** (**LEA** R6 #-2)

FETCH

in: enable_fetch = 1 ; pc = **3001** ; npc = **3002** ;

out: instrmem_rd = 1 ;

DECODER

in: Instr_dout = IMem[**3001**] = **1422** ; npc_in = **3002** ;

enable_decoder = 1 ;

out: npc_out = **3002**; IR = **1422** ; W_Control = **0** (aluout) ;

alu_control = **0**; pcselect1 = 0; pcselect2 = 0; op2select = **0** (imm5)

=> E_Control = **[6'b00_0000]**

EXECUTE

in: E_Control = **[6'b00_0000]**; W_Control_in = 0 ; npc_in = **3002** ;

IR = **1422** ; enable_execute = 1 ;

VSR1 = **R0 = 0** ; VSR2 = **R2 = xxx** ;

out: sr1 = **0**; sr2 = **2** ;

dr = **2** ; W_Control_out = 0 ; aluout = **VSR1 + sxt(Imm5) = R0 + 2 = 2**

WRITEBACK

in: dr = **2** ; W_Control = 0 ; aluout = **2** ; enable_writeback = 1 ;

out: RegFile[dr] = aluout = 2 => **R2 = 2**

UPDATEPC

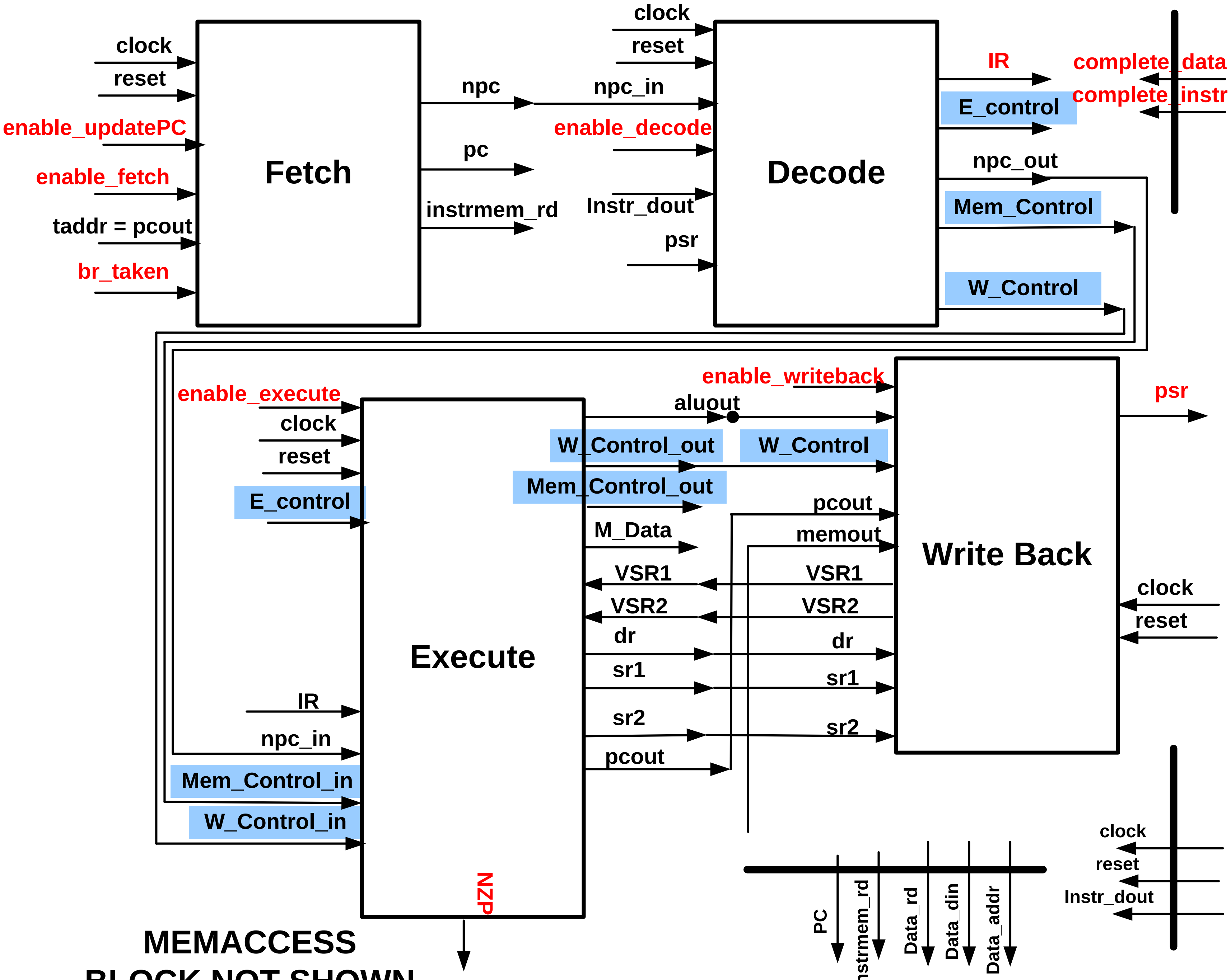
in: enable_updatePC = 1 ; **out:** pc = **3002** ; npc = **3003**

Detailed Timing Example

```
@3000: 5020 (AND R0 R0 #0);    @3001: 1422 (ADD R2 R0 #2)
@3002: 1280 (ADD R1 R2 R0);    @3003: EDFE (LEA R6 #-2)

FETCH
in: enable_fetch = 1 ; pc = 3002 ; npc = 3003 ;
out: instrmem_rd = 1 ;

DECODER
in: Instr_dout = IMem[3002] = 1280 ;  npc_in  = 3003 ;
    enable_decoder = 1 ;
out: npc_out = 3003; IR = 1280 ; W_Control = 0 (aluout) ;
     alu_control = 0; pcselect1 = 0; pcselect2 = 0; op2select = 1 (VSR2)
     => E_Control = [6'b00_0001]

EXECUTE
in: E_Control = [6'b00_0001]; W_Control_in = 0 ; npc_in = 3003 ;
    IR = 1280 ; enable_execute = 1 ;
    VSR1 = R2 = 2 ; VSR2 = R0 = 0 ;
out: sr1 = 2; sr2 = 0 ;
    dr = 1 ; W_Control_out = 0 ; aluout = VSR1 + VSR2 = 2 + 0 = 2

WRITEBACK
in: dr = 1 ; W_Control = 0 ; aluout = 2 ; enable_writeback = 1 ;
out: RegFile[dr] = aluout = 2 => R1 = 2

UPDATEPC
in: enable_updatePC = 1 ; out: pc = 3003 ; npc = 3004
```

Detailed Timing Example

```
@3000: 5020 (AND R0 R0 #0);   @3001: 1422 (ADD R2 R0 #2)
@3002: 1280 (ADD R1 R2 R0);   @3003: EDFE (LEA R6 #-2)

FETCH
in: enable_fetch = 1 ; pc = 3003 ; npc = 3004 ;
out: instrmem_rd = 1 ;

DECODER
in: Instr_dout = IMem[3003] = EDFE;  npc_in  = 3004 ;
    enable_decoder = 1 ;
out: npc_out = 3004 ; IR = EDFE; W_Control = 2 (pcout) ;
     alu_control=0; pcselect1 = 1(offset9); pcselect2 = 1(npc); op2select = 0
     => E_Control = [6'b00_0110]

EXECUTE
in: E_Control = [6'b00_0110]; W_Control_in = 2 ; npc_in = 3004 ;
    IR = EDFE; enable_execute = 1 ;
    VSR1 = R7 = xxx ; VSR2 = R6 = xxx ;
out: sr1 = 7; sr2 = 6 ;
    dr = 6 ; W_Control_out=2 ; pcout = npc + sxt(offset9) = 3004 + -2 = 3002

WRITEBACK
in: dr = 6 ; W_Control = 2 ; pcout = 3002 ; enable_writeback = 1 ;
out: RegFile[dr] = pcout = 3002 => R6 = 3002

UPDATEPC
in: enable_updatePC = 1 ; out: pc = 3004 ; npc = 3005
```



MEMACCESS
BLOCK NOT SHOWN

Testing Considerations

- Where Do you start? Understand:
  - Inputs and Outputs to the design & their constraints
  - Timing of instructions (5 cycles ignoring `complete_instr`)
    - Controller timing

- Basic Checks
  - ALWAYS INITIALIZE REGISTERS
  - Reset Behavior (Each block has a particular behavior)
  - Completion of instructions and time taken
  - Operational Correctness i.e. results are right
  - Register File correctness

- Further thoughts
  - Smart use of data values: use inputs that give you maximum information
  - Sequence of instructions: ADD→NOT→AND→ADD with dr in one instruction being the sr1/2 in the next.