ECE745: PROJECT1: LC3 INSTRUCTION SET ARCHITECTUE

Department of Electrical and Computer Engineering
NC STATE UNIVERSITY

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).

LC-3 Instruction Set Architecture

- 16-bit wide memory
  - Each word is 2-bytes wide
  - The 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 entry in 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 = `InstrMem[PC]`
- Addressing mode
  - Immediate (with sign extension) `([DR] ← [SR1] & signextend(Imm))`
  - Register (SR1 SR2) `([DR] ← [SR1] + [SR2])`
  - Memory:
    - PC relative `Mem[PC + signextend(Offset)]`
    - Indirect `Mem[Mem[PC + signextend(Offset)]]`
    - Base register +offset
      `Mem[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, Imm ;`**`IR[5]=1`**`; ([DR]←[SR1]`**`&`**` Imm)`
- ADD `IR[15:12] = 0001`
  - `ADD DR, SR1, SR2 ;`**`IR[5]=0`**`; ([DR]←[SR1]`**`+`**`[SR2])`
  - `ADD DR, SR1, Imm ;`**`IR[5]=1`**`; ([DR]←[SR1]`**`+`**` Imm)`
- NOT `IR[15:12] = 1001`
  - `NOT DR, SR1                 ([DR] ← ~[SR1])`

Note: `Imm` is going to be sign extended to 16 bits for all computation.

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]` $\leftarrow$ `PC+1+Offset)`

    `Offset = sxt(PCOffset9)`
- Load/Store (PC relative addressing mode)
  - LD DR, `Offset` `([DR]` $\leftarrow$ **`MEM`**`[PC+1+Offset])`
  - ST SR, `Offset` `(`**`MEM`**`[PC+1+ Offset]` $\leftarrow$ `[SR])`

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

    `([DR]` $\leftarrow$ **`MEM`**`[BaseR+Offset])`
  - STR SR, BaseR, `Offset`

    `(`**`MEM`**`[BaseR+Offset]` $\leftarrow$ `[SR])`

  `Offset = sxt(PCOffset6)`

Memory Operations

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

    `(DR ←  `**`MEM`**`[`**`MEM`**`[PC+1+Offset]])`
  - STI SR, `Offset`

    `(`**`MEM`**`[`**`MEM`**`[PC+1+Offset]] ← SR)`

`Offset = sxt(PCOffset9)`

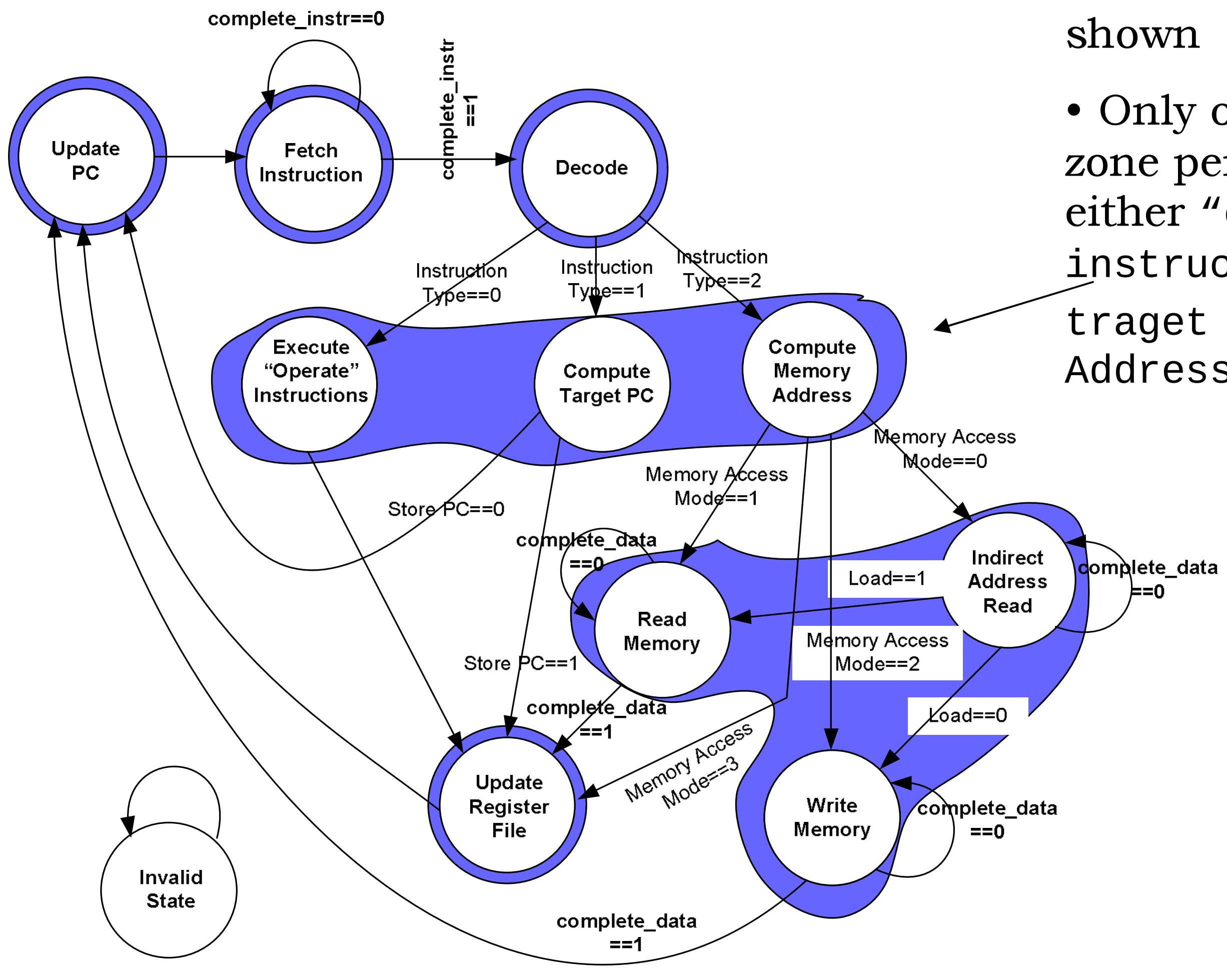
Project1 Specifics

- Subset of instructions
  - ALU (ADD, NOT, AND) ; LEA
  - All instructions take 5 clock cycles
- Block level inputs and outputs
  - 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 like control and data dependence .. FOR NOW !!

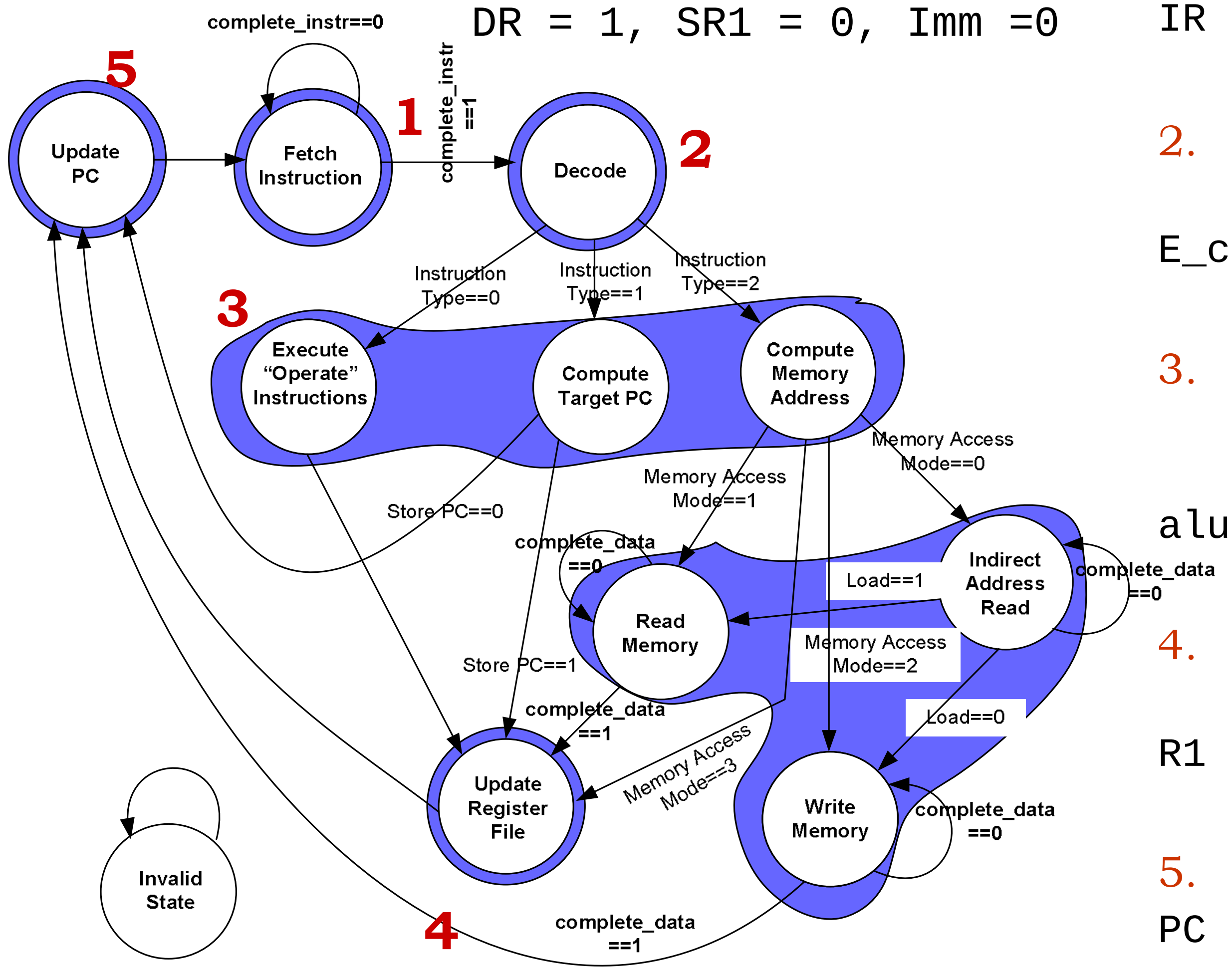
LC-3 Components (Base States)



- 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”

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

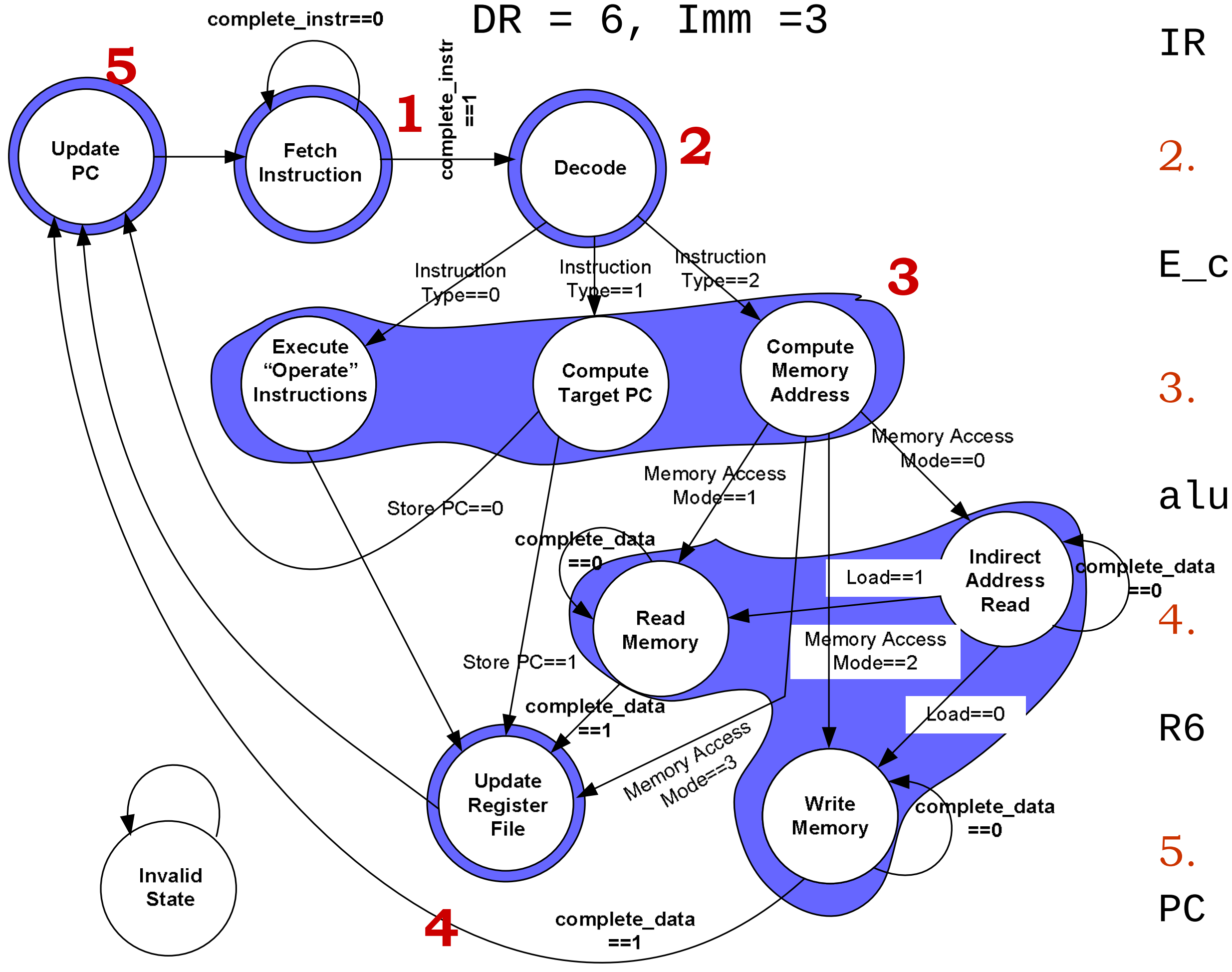
```
R1 = aluout
```

5. PC incremented

```
PC = PC + 1
```

LEA Base States

```
Example:
IR = EC08 (LEA R6,#8)
DR = 6, Imm =3
```



1. Fetch Unit loads instruction from memory

`IR = IMem[PC] = EC08`

2. Decode Unit determines the operands

`E_control, W_Control etc.`

3. Compute Memory Address with Offset

`aluout = PC+1+8`

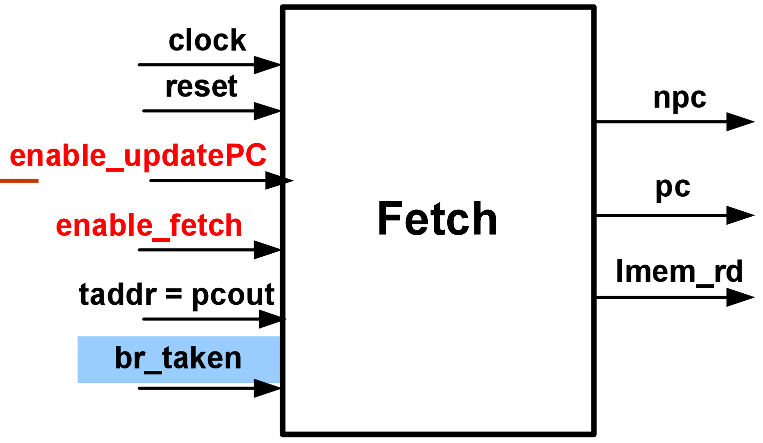
4. Write Memory Address to Reg File

`R6 = aluout`

5. PC incremented

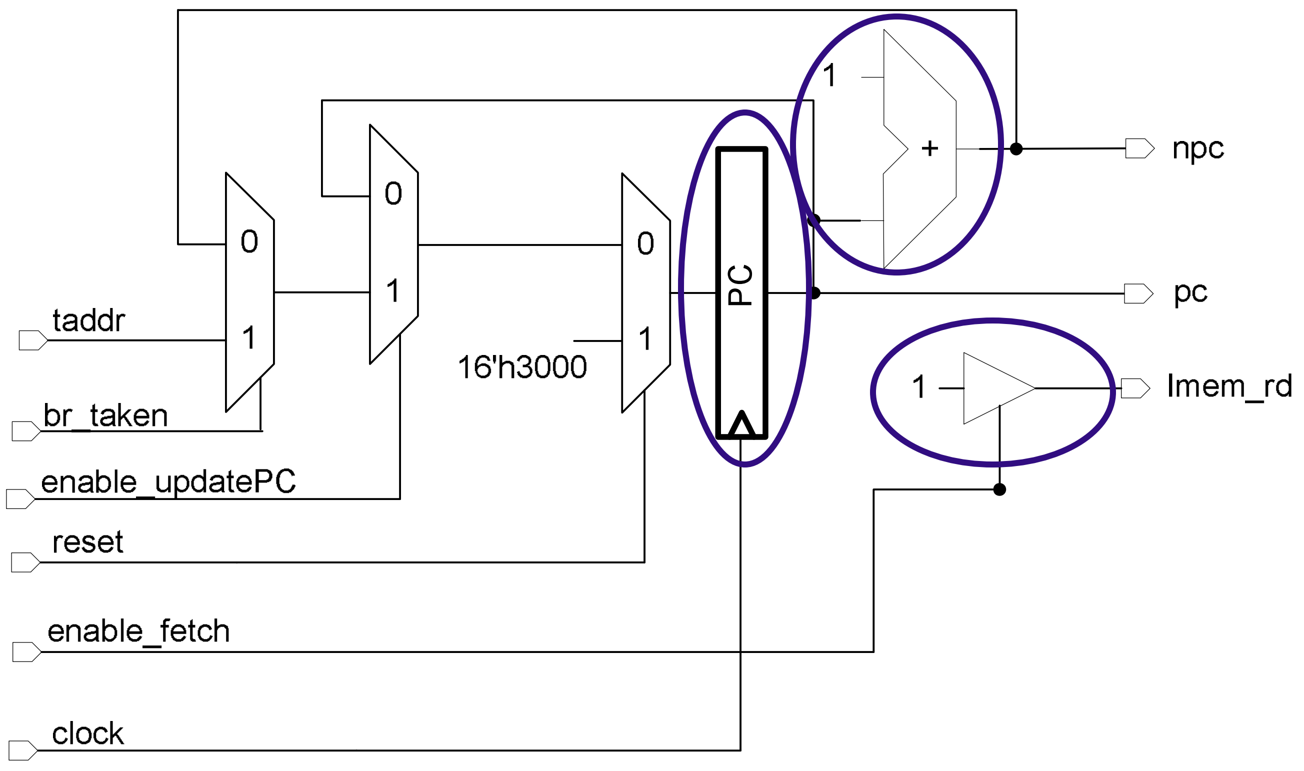
`PC = PC + 1`

LC3 Internals FETCH

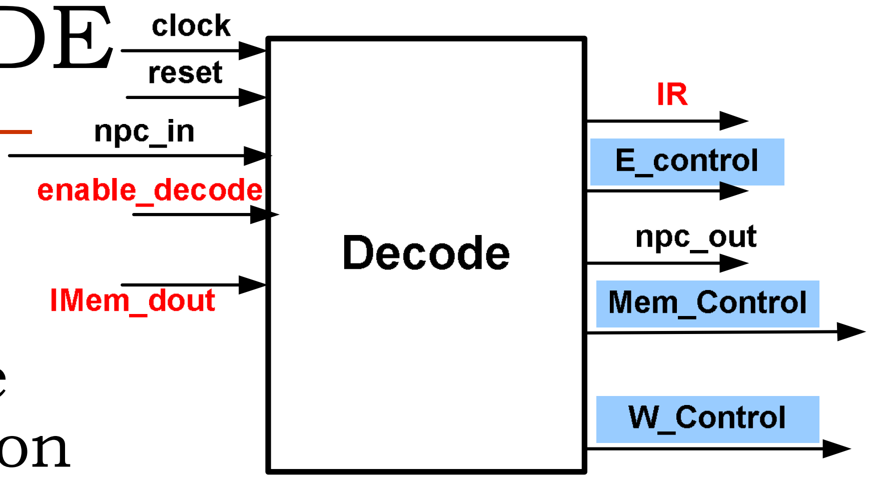


- On reset: `pc =3000; npc =3001;`
- Signals of Importance
  - `enable_fetch` and `enable_updatePC`
  - `PC` = Program Counter of instruction being fetched
  - `npc = PC + 1`
  - `Imem_rd` = enable for Instruction Mem Read
- Ignore
  - `br_taken` = if 1 `PC = taddr` else `PC = PC+1`
  - `taddr` = new PC if branch is taken
- if `enable_fetch=1 Imem_rd=1` asynchronously else `Imem_rd=Z` (HIGH IMPEDANCE)
- `PC, npc, Imem_rd` updated only when `enable_updatePC=1`

LC3 Internals FETCH



LC3 Internals DECODE



- Creates relevant control signals using `IMem_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 `Imem_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

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)

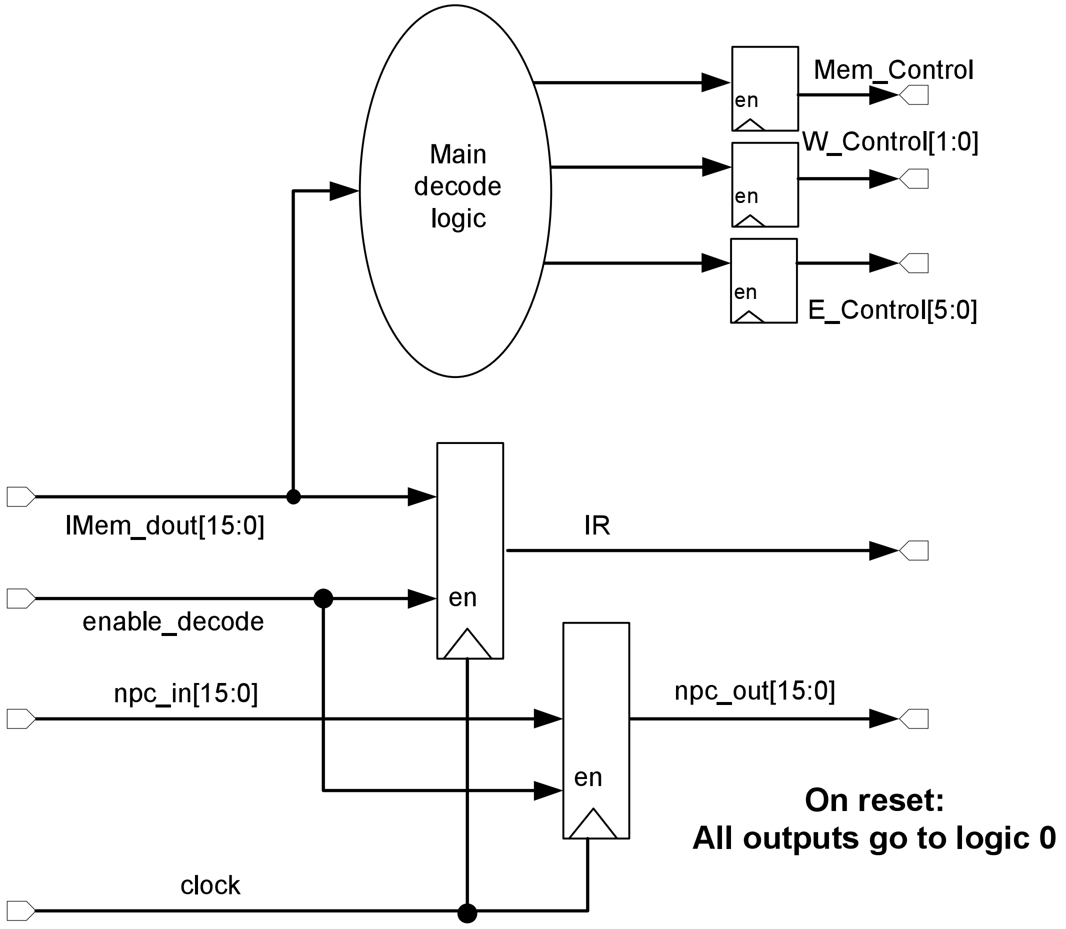
Operation	mode	**W_Control**
ADD	0	**0(aluout)**
	1	**0(aluout)**
AND	0	**0(aluout)**
	1	**0(aluout)**
NOT		**0(aluout)**
BR		0
JMP		0
LD		**1(memout)**
LDR		**1(memout)**
LDI		**1(memout)**
LEA		**2(pcout)**
ST		0
STR		0
STI		0

LC3 Internals DECODE

- `E_Control:{alu_control, pcselect1, pcselect2, op2select}`

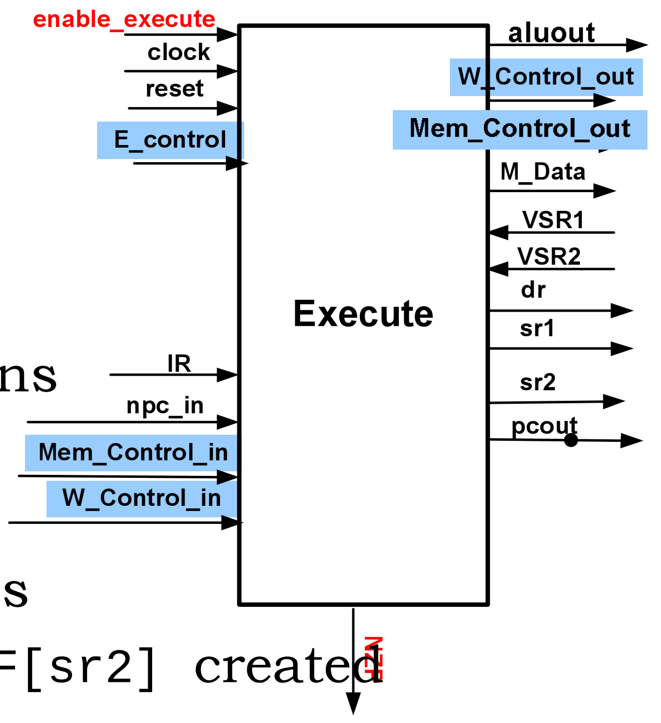
		E_Control			
IR[15:12]	IR[5]	**alu_control [2]**	**pcselect1 [2]**	**pcselect2 [1]**	**op2select [1]**
ADD	0	**0**	-	-	VSR2 **(1)**
	1	**0**	-	-	imm5 **(0)**
AND	0	**1**	-	-	VSR2 **(1)**
	1	**1**	-	-	imm5 **(0)**
NOT		**2**	-	-	-
BR		-	offset9 **(1)**	npc **(1)**	-
JMP		-	0 **(3)**	VSR1 **(0)**	-
LD		-	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 synchrously `W_control_in`
- Ignore
  - `M_Data` = RF value to be written to memory
  - `Mem_control_in, NZP, Mem_control_out`

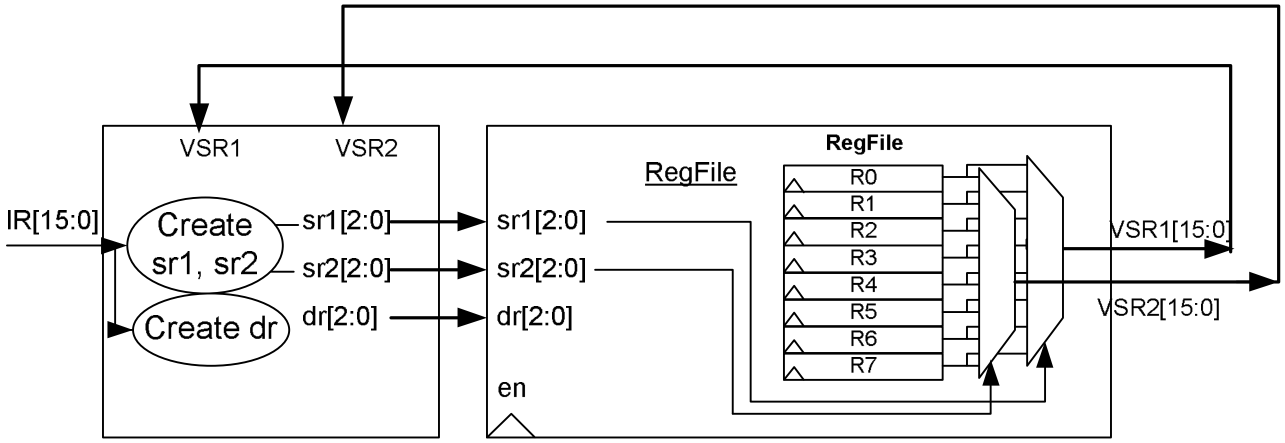
LC3 Internals EXECUTE

([DR]←[SR1] **aluop** [SR2])

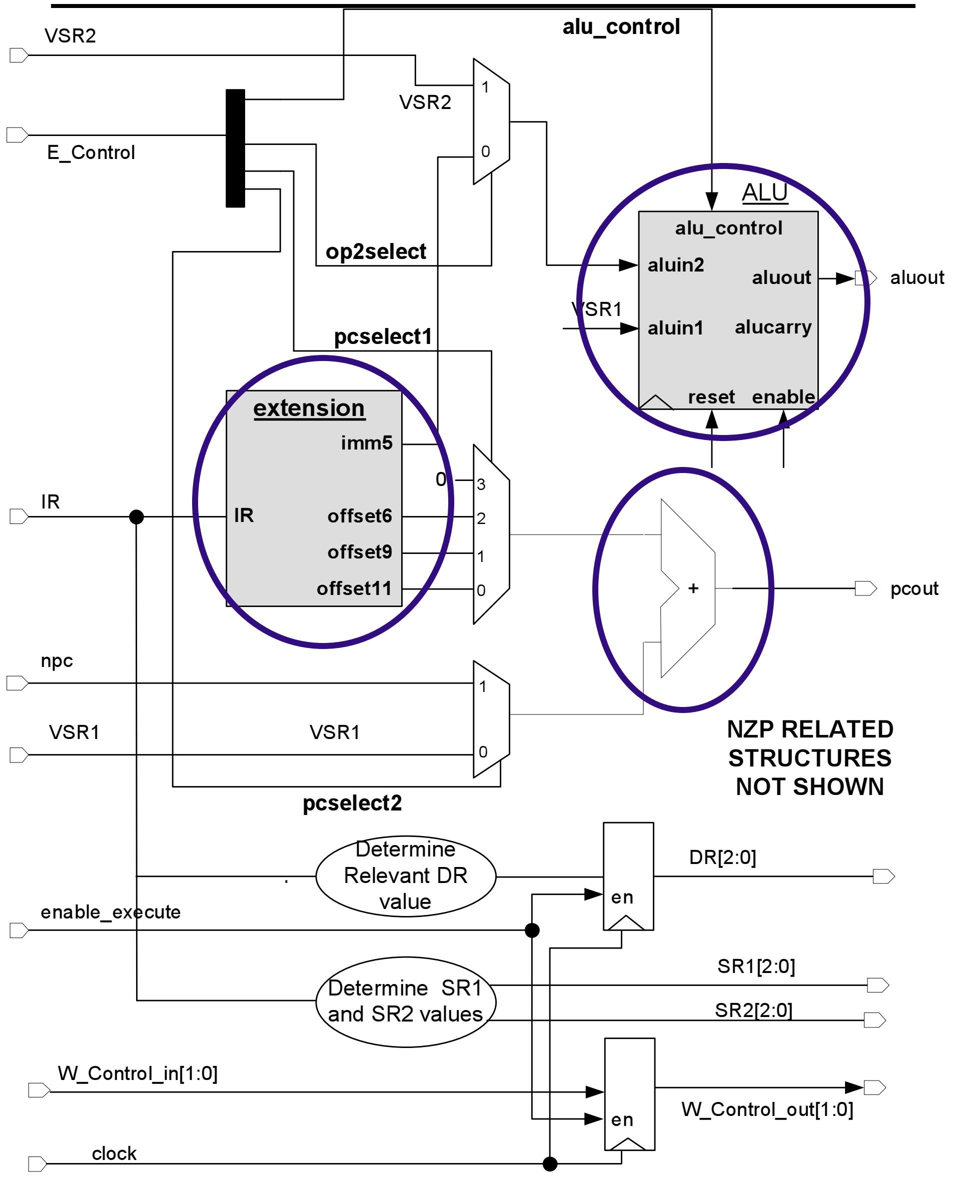
([DR]←[SR1] **aluop** **sxt**(Imm))

LEA ([DR] ← PC+1+**sxt**(PCOffset9))

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



LC3 Internals EXECUTE



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**

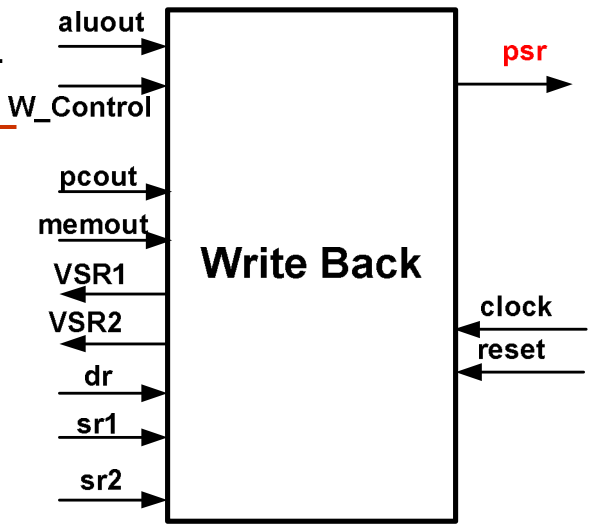
ALU Operation Instructions

	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		

Memory Operations

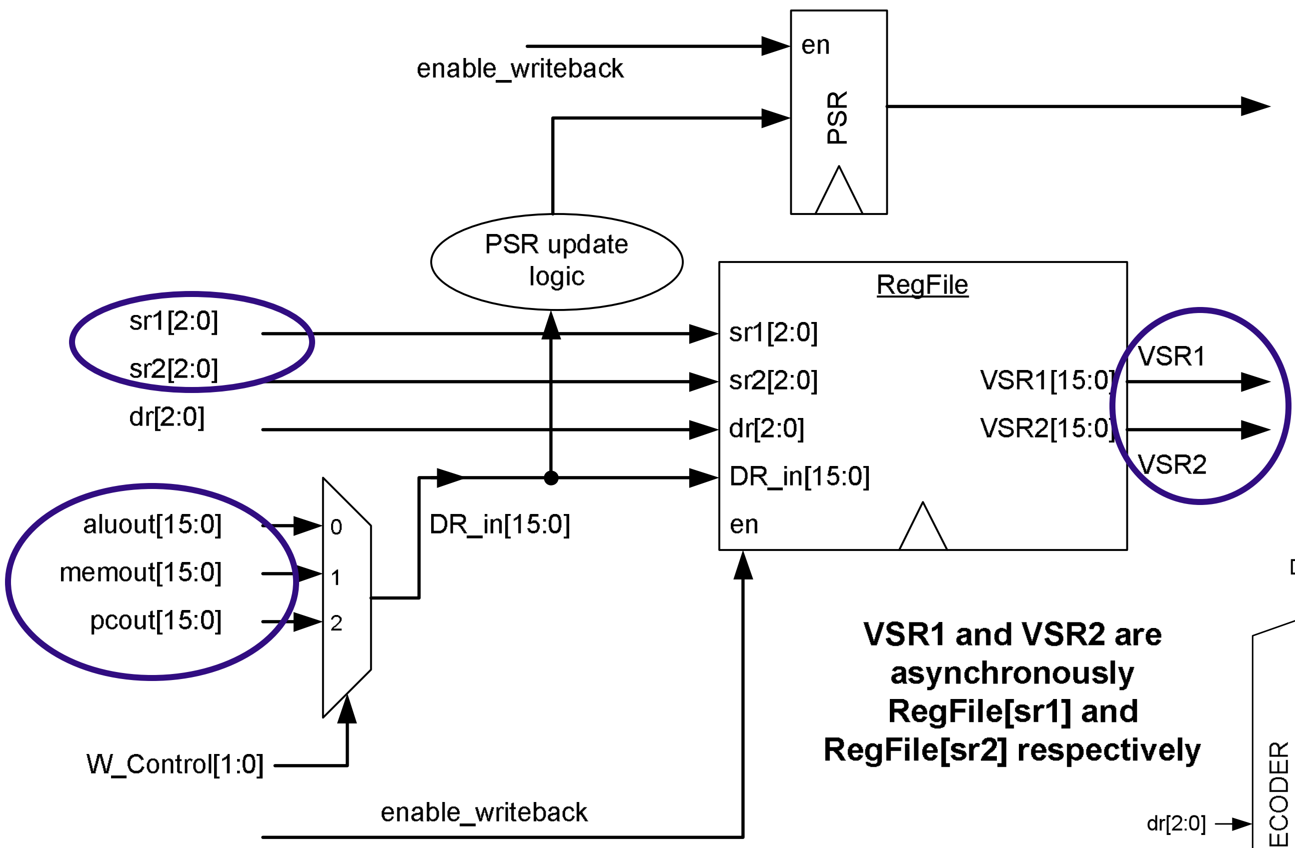
	15 12	11 9	8 6	5 4 3 2 0
LD	0 0 **1 0**	DR	PCoffset9	
LDR	0 1 **1 0**	DR	BaseR	Offset6
LDI	1 0 **1 0**	DR	PCoffset9	
LEA	1 1 **1 0**	DR	PCoffset9	
ST	0 0 **1 1**	SR	PCoffset9	
STR	0 1 **1 1**	SR	BaseR	Offset6
STI	1 0 **1 1**	SR	PCoffset9	

LC3 Internals WRITEBACK

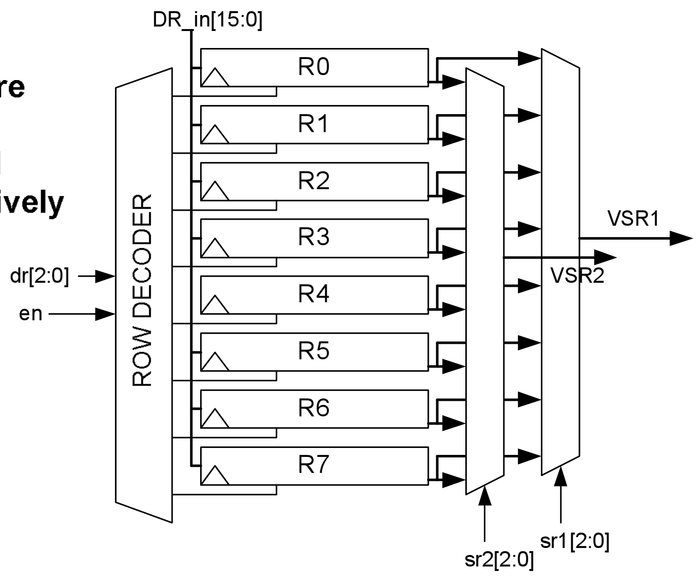


- Contains register file (RF/RegFile)
  - Addressed using `sr1, sr2` and `dr`
  - Writes either `aluout, pcout OR memout` based on `W_control_in` value
  - Synchronous Writes to RF with `dr`
    `RegFile[dr] = DR_in`
  - Asynchronous Reads from RF using `sr1&sr2`
- Important Signals:
  - `enable_writeback`: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

LC3 Internals WRITEBACK



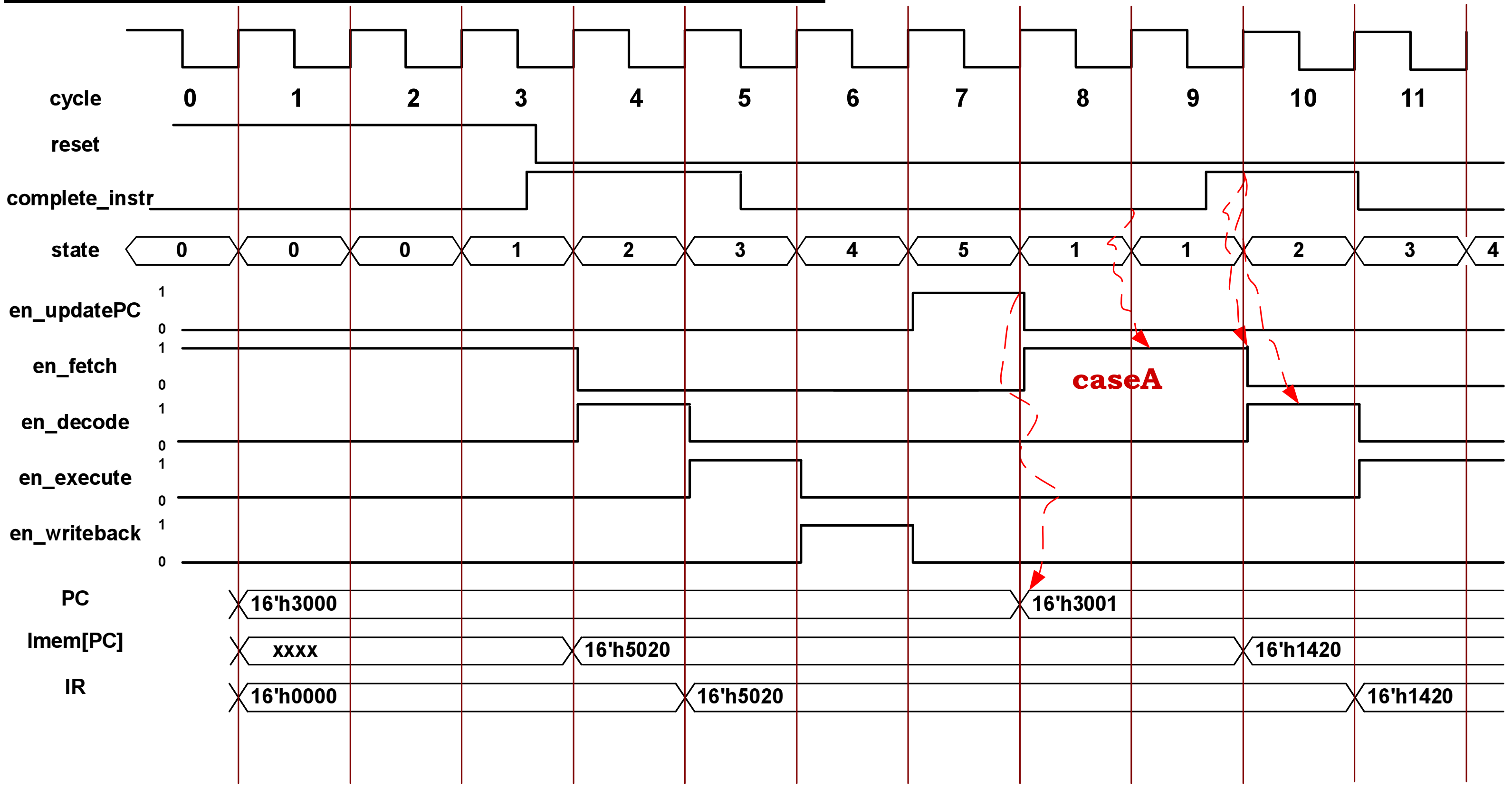
VSR1 and VSR2 are asynchronously RegFile[sr1] and RegFile[sr2] respectively



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: 1820 (ADD R1 R2 R0); **@3003: EC03 (LEA** R6 #-2)

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

DECODER
in: IMem_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: 1820 (**ADD** R1 R2 R0); **@3003: EC03** (**LEA** R6 #-2)

FETCH
in: enable_fetch = 1 ; pc = **3001** ; npc = **3002** ;
out: IMem_rd = 1 ;

DECODER
in: IMem_dout = IMem[3001] = **1420** ; npc_in = **3002** ;
enable_decoder = 1 ;
out: npc_out = **3002**; IR = **1420** ; W_Control = **0** (aluout) ;
alu_control = **0**; 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 = **3002** ;
IR = **1420** ; enable_execute = 1 ;
VSR1 = **R0 = 0** ; VSR2 = **R0 = 0** ;
out: sr1 = 0; sr2 = 0 ;
dr = 0 ; W_Control_out = 0 ; aluout = **VSR1 + sxt(Imm5) = 0 + 2 = 2**

WRITEBACK
in: dr = **2** ; W_Control = 0 ; aluout = 0 ; 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: 1820 (ADD R1 R2 R0);   @3003: EC03 (LEA R6 #-2)

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

DECODER
in: IMem_dout = IMem[3001] = 1820 ;  npc_in  = 3003 ;
    enable_decoder = 1 ;
out: npc_out = 3003; IR = 1820 ; W_Control = 0 (aluout) ;
     alu_control = 0; pcselect1 = 0; pcselect2 = 0; op2select = 1 (VSR2)
     => E_Control = [6'b01_0001]

EXECUTE
in: E_Control = [6'b01_0001]; W_Control_in = 0 ; npc_in = 3003 ;
    IR = 1820 ; enable_execute = 1 ;
    VSR1 = R2 = 2 ; VSR2 = R0 = 0 ;
out: sr1 = 0; sr2 = 0 ;
    dr = 0 ; W_Control_out = 0 ; aluout = VSR1 + VSR2 = 2 + 0 = 2

WRITEBACK
in: dr = 1 ; W_Control = 0 ; aluout = 0 ; enable_writeback = 1 ;
out: RegFile[dr] = aluout = 0 => 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: 1820 (ADD R1 R2 R0);   @3003: EDFE (LEA R6 #-2)
```

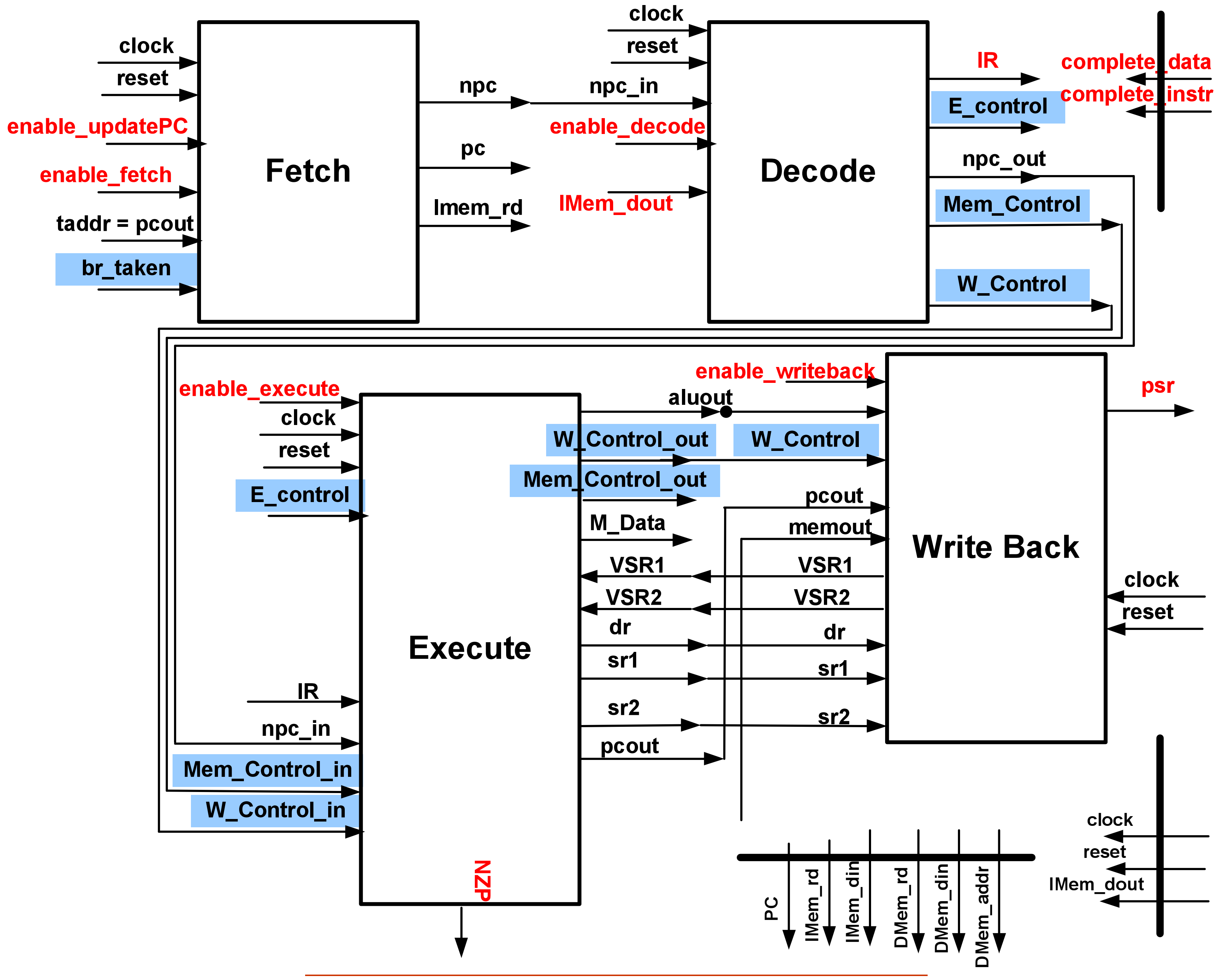
```
FETCH
in: enable_fetch = 1 ; pc = 3003 ; npc = 3004 ;
out: IMem_rd = 1 ;

DECODER
in: IMem_dout = IMem[3001] = 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'b01_0110]

EXECUTE
in: E_Control = [6'b01_0110]; W_Control_in = 2 ; npc_in = 3004 ;
    IR = EDFE; enable_execute = 1 ;
    VSR1 = R0 = 0 ; VSR2 = R0 = 0 ;
out: sr1 = 0; sr2 = 0 ;
    dr = 6 ; W_Control_out=2 ; aluout = npc + sxt(offset9) = 3004 + -2 = 3002

WRITEBACK
in: dr = 6 ; W_Control = 0 ; aluout = 0 ; enable_writeback = 1 ;
out: RegFile[dr] = aluout = 0 => R6 = 3002

UPDATEPC
in: enable_updatePC = 1 ; out: pc = 3002 ; npc = 3003
```



Fetch
clock
reset
enable_updatePC
enable_fetch
taddr = pcout
br_taken
npc
pc
Imem_rd
Decode
clock
reset
npc_in
enable_decode
IMem_dout
IR
E_control
npc_out
Mem_Control
W_Control
complete_data
complete_instr
Execute
enable_execute
clock
reset
E_control
IR
npc_in
Mem_Control_in
W_Control_in
NZP
aluout
W_Control_out
Mem_Control_out
M_Data
VSR1
VSR2
dr
sr1
sr2
pcout
Write Back
enable_writeback
W_Control
pcout
memout
VSR1
VSR2
dr
sr1
sr2
psr
clock
reset
PC
IMem_rd
IMem_din
DMem_rd
DMem_din
DMem_addr
clock
reset
IMem_dout

Testing Considerations

- Where Do you start?
  - Inputs and Outputs to the design & their constraints
  - Timing of instructions (5 cycles)
  - Understand controller timing
- Basic Checks
  - 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 issues
  - 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.