CG2028 Lecture 4 : Single Cycle Processor Design

Ankit Srivastava ECE, NUS

Slides from Dr Rajesh Panicker

Acknowledgement / References:

- Text by Patterson and Hennessey
- Text and by Harris and Harris

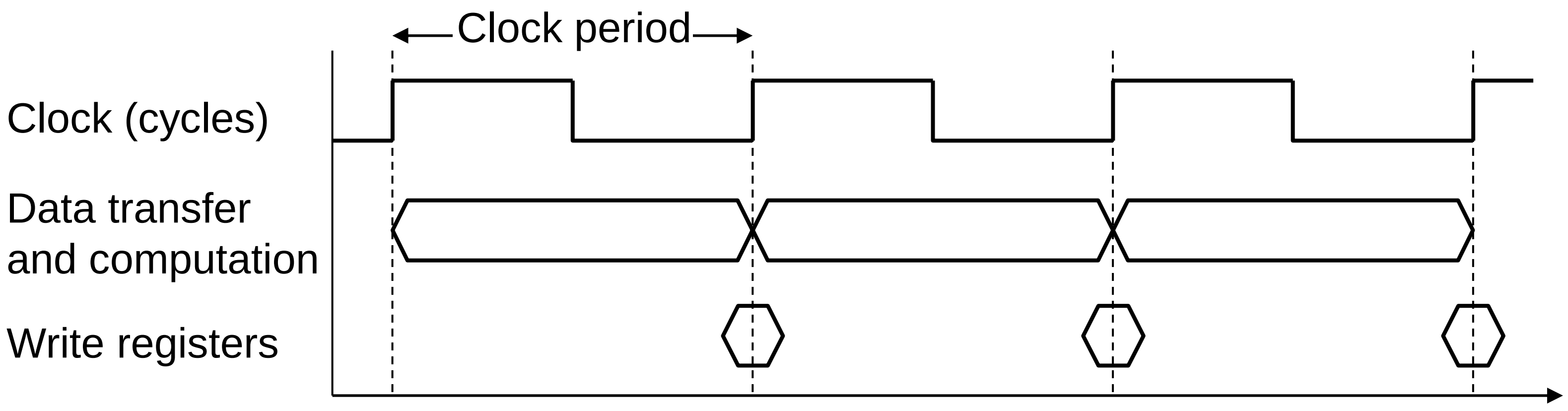
Contents

- ARM-like instruction encoding
  - The design and encoding in this chapter is not compliant with any version of ARM Architecture, though it is closer to ARMv3 than ARMv7M covered in chapter 3
  - We will be looking at a simplified version

    (different from all books/references, but somewhat similar to the one in Harris and Harris)
- Microarchitecture : Datapath design
- Microarchitecture : Controller design
- Extensions for other formats / instructions

CPU Clocking

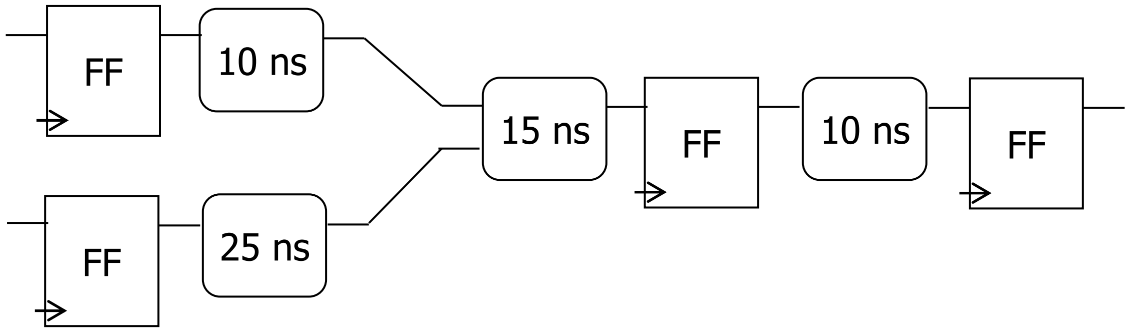
- Operation of digital hardware governed by a constant-rate clock



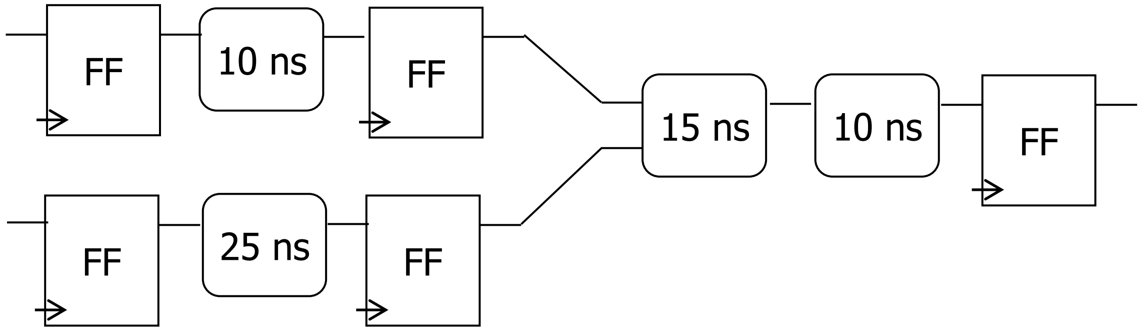
- Clock period: duration of a clock cycle
  - e.g., 500ps = 0.5ns = 500×10^{-12}s
- Clock frequency (rate): cycles per second
  - e.g., 2.0GHz = 2000MHz = 2.0×10^{9}Hz

Critical Path

- Critical path = combinational path with maximum delay
- Determines the max. clock
  - assuming FFs have negligible setup time and propagation delay



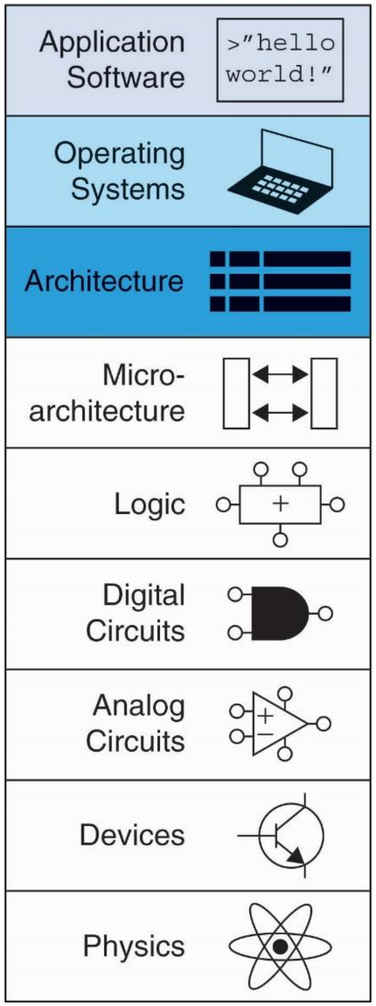
Max clock
= 1/Critical path delay
= 1/40 ns = 25 MHz



Max clock
= 1/Critical path delay
= 1/25 ns = 40 MHz

Architecture vs Microarchitecture

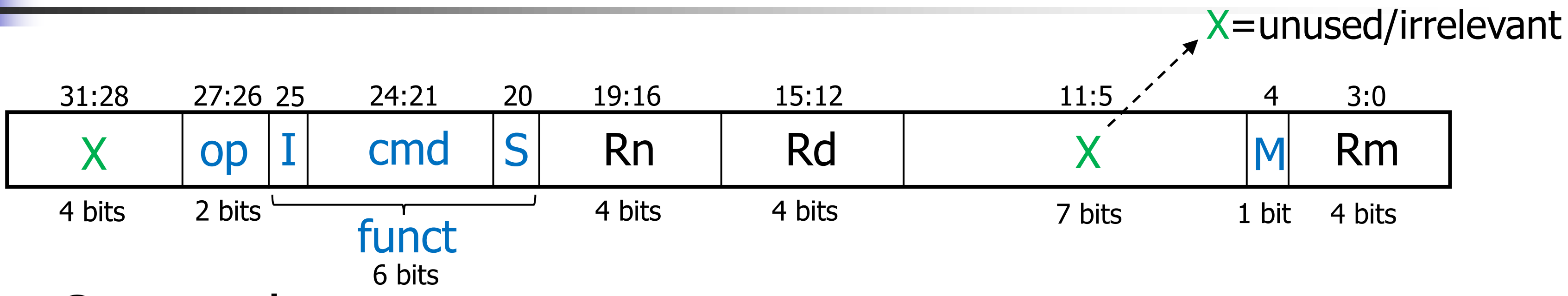
- Architecture: programmer's view of computer
  - Defined by instructions & operand locations
  - Assembly language: human-readable format of instructions
  - Machine language: computer-readable format (1's and 0's)
  - Assembly language -> Machine language conversion is done by the assembler
    - one to one correspondence
      (except for pseudo-instructions)
- Microarchitecture: digital designer's view of the computer
  - How to implement an architecture in hardware
  - Different designs possible to execute the same code
    - Price-performance trade-off



Architecture : Instruction Formats

- We will be looking at 3 instruction formats, with restrictions
  - Data-processing (DP) - ADD, SUB, AND, ORR
    - 2 variants – register and immediate *Operand2*
    - Other DP operations not supported
  - Memory - LDR, STR
    - Only offset mode
    - Magnitude of the offset is 8 bits
  - Branch - B, BEQ
    - Only unconditional and EQ
    - Magnitude of the offset from PC+4 is 8 bits
    - No reads or writes to/from PC/R15 other than by branch instruction

DP Register Operand2 Format



- Operands
  - *Rn* : first source register
  - *Rm* : second source register
  - *Rd* : destination register
- Control fields
  - op : the operation code or opcode
    - op = 0b00 for data-processing (DP) instructions
  - funct is composed of cmd, I-bit, and S-bit
    - cmd = 0b0000 for AND, 0b0010 for SUB, 0b0100 for ADD, 0b1100 for ORR
    - I = immediate = **0b0** for **register** Operand2
    - S = set flags = 0b1 if the suffix S is specified, for example, ADDS, ANDS
  - M = 0b0

```
OP{S} {Rd,} Rn, Rm
OP => ADD, SUB, AND, ORR
```

When `Rd` is not specified in assembly, `Rd` = `Rn`. It is not left 'empty' in machine code

DP Register Operand2 Example

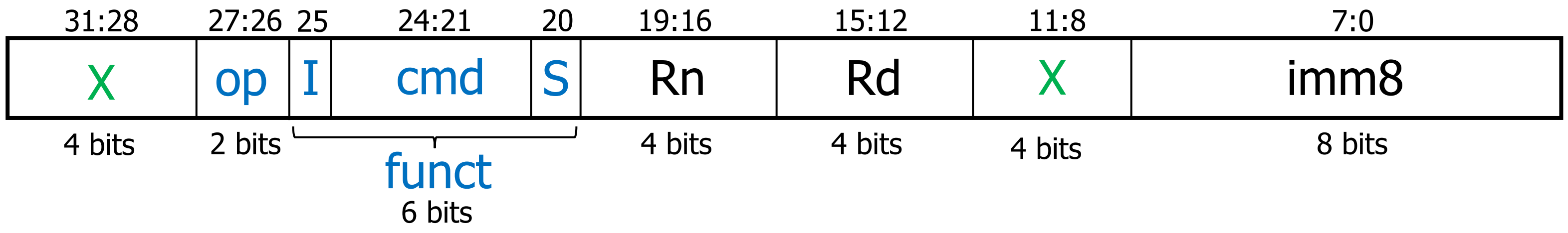
- ADDS R2, R3, R5
  - op = 0b00 for DP instructions
  - cmd = 0b0100 for ADD
  - *Operand2* is a register so I=0b0
  - Sets the flags, so S=0b1
  - *Rd* = 2, *Rn* = 3
  - *Rm* = 5
  - Assume X's are 0s

`<OP Rd, Rn, Rm>`

31:28	27:26	25	24:21	20	19:16	15:12	11:5	4	3:0
X	op	I	cmd	S	Rn	Rd	X	M	Rm
0000	00	0	0100	1	0011	0010	0000000	0	0101

ADDS R2, R3, R5 => 0x00932005

DP Immediate Operand2 Format



```
OP{S} {Rd,} Rn, #imm8
OP => ADD, SUB, AND, ORR
```

- Operands
  - *Rn* : first source register
  - *imm8* : 8-bit unsigned immediate
  - *Rd* : destination register
- Control fields
  - op : the operation code or opcode
    - op = 0b00 for data-processing (DP) instructions
  - funct is composed of cmd, I-bit, and S-bit
    - cmd = 0b0000 for AND, 0b0010 for SUB, 0b0100 for ADD, 0b1100 for ORR
    - I = immediate = **0b1** for **immediate** Operand2
    - S = set flags = 0b1 if the suffix S is specified, for example, ADDS, ANDS

DP Immediate Operand2 Example

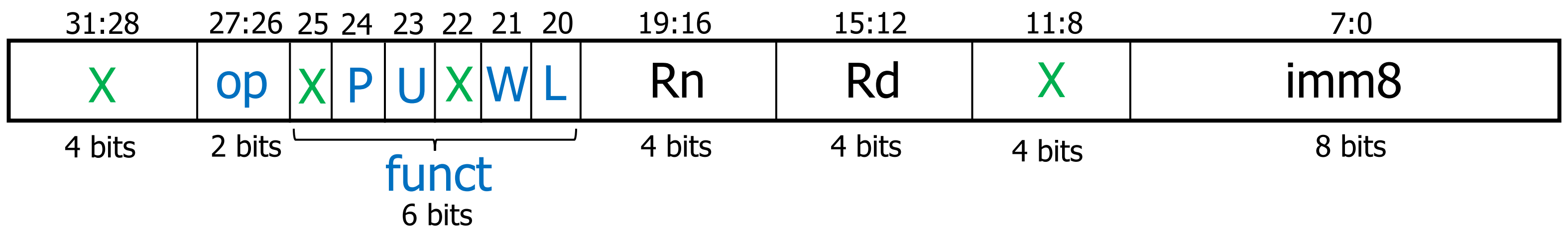
- SUB R2, R3, #0xAB
  - op = 0b00 for DP instructions
  - cmd = 0b0010 for SUB
  - *Operand2* is an immediate so I=0b1
  - Doesn’t set the flags, so S=0b0
  - *Rd* = 2, *Rn* = 3
  - *imm8* = 0xAB
  - Assume X’s are 0s

`<OP Rd, Rn, #imm8>`

31:28	27:26	25	24:21	20	19:16	15:12	11:8	7:0
X	op	I	cmd	S	Rn	Rd	X	imm8
0000	00	1	0010	0	0011	0010	0000	10101011

SUB R2, R3, #0xAB => 0x024320AB

Memory Instruction Format



```
OP Rd, [Rn, #±imm8]
OP => LDR, STR
```

- Encodes LDR, STR
  - op : 0b01 for memory instructions
  - funct : 6 control bits
  - U : Add
    - 0b1 -> offset is positive, i.e., effective address = *Rn* + *imm8*
    - 0b0 -> the offset is negative, i.e., effective address = *Rn* – *imm8*
  - L = 0b1 for load; 0b0 for store
  - PW = 0b10
  - *Rn* : base register
  - *Rd* : destination (load), source (store)
  - *imm8* : magnitude of offset

Memory Instruction Example

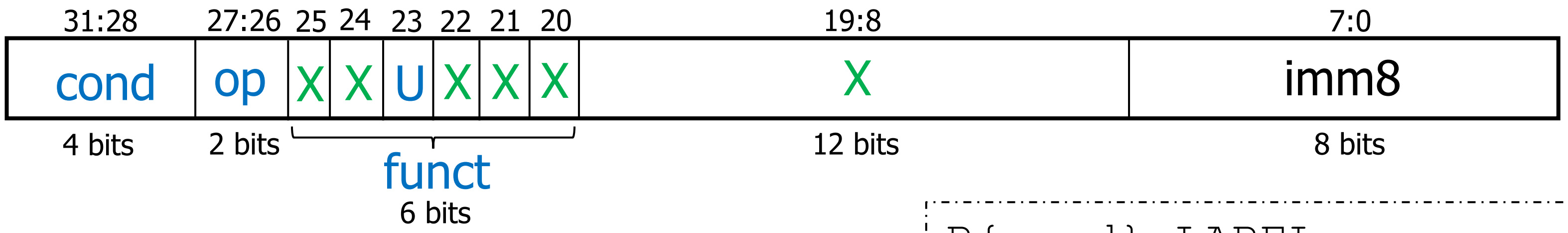
- STR R11, [R5, #-26]
  - Operation: mem[R5-26] <= R11;
  - op = 0b01 for memory instruction
  - U = 0b0 (offset is negative, subtract)
  - L = 0b0 (store)
  - *Rd* = 11, *Rn* = 5, *imm8* = 26 = 0x1A

`<OP Rd, [Rn, #±imm8]>`

31:28	27:26	25	24	23	22	21	20	19:16	15:12	11:8	7:0
X	op	X	P	U	X	W	L	Rn	Rd	X	imm8
0000	01	0	1	0	0	0	0	0101	1011	0000	00011010

STR R11, [R5, #-26] => 0x0505B01A

Branch Instruction Format



```
B{cond} LABEL
LABEL encoded as #±imm8
```

- Encodes B{cond}
  - cond : condition to be true for the branch to be taken
    - EQ = 0b0000
    - AL (always a.k.a unconditional) = 0b1110
  - op = 0b10 for branch instructions
  - *imm8* : 8-bit immediate encoding Branch Target Address (BTA)
    - BTA = address corresponding to LABEL = Next PC when branch taken
    - *imm8* = # of bytes BTA is away from current PC+4
  - U : add
    - 0b1 -> BTA = PC+4+*imm8*; 0b0 -> BTA = PC+4-*imm8*

Branch Instruction Example

```
0x8040 TEST: LDR  R5, [R0, #4] ←BTA
0x8044       STR  R5, [R1, #1]
0x8048       ADD  R3, R3, #1
0x804C       MOV  R5, R4
0x8050       B    TEST         ← PC
0x8054       LDR  R3, [R1]     ← PC+4
0x8058       SUB  R4, R3, #9
```

- PC = 0x8050
- PC+4 = 0x8054
- `BTA = TEST` = 0x8040
- offset = 0x8040-0x8054
  = -0x14,
  encoded as U = 0b0
  imm8 = 0x14

- B TEST
  - cond = 0b1110 for unconditional branch
  - op = 0b10 for branch instructions
  - U = 0b0 (subtract); *imm8* = 0x14

31:28	27:26	25	24	23	22	21	20	19:8	7:0
cond	op	X	X	U	X	X	X	X	imm8
1110	10	0	0	0	0	0	0	000000000000	00010100

B TEST => 0xE8000014

Review : Instruction Formats

31:28	27:26	25	24:21	20	19:16	15:12	11:5	4	3:0
X	op	I	cmd	S	Rn	Rd	X	M	Rm
4 bits	2 bits	funct 6 bits			4 bits	4 bits	7 bits	1 bit	4 bits

DP register
```
OP{S} Rd, Rn, Rm
OP => ADD, AND,..
```

31:28	27:26	25	24:21	20	19:16	15:12	11:8	7:0
X	op	I	cmd	S	Rn	Rd	X	imm8
4 bits	2 bits	funct 6 bits			4 bits	4 bits	4 bits	8 bits

DP immediate
```
OP{S} Rd, Rn, #imm8
OP => ADD, AND,..
```

31:28	27:26	25	24	23	22	21	20	19:16	15:12	11:8	7:0
X	op	X	P	U	X	W	L	Rn	Rd	X	imm8
4 bits	2 bits	funct 6 bits						4 bits	4 bits	4 bits	8 bits

Memory
```
OP Rd, [Rn, #±imm8]
OP => LDR, STR
```

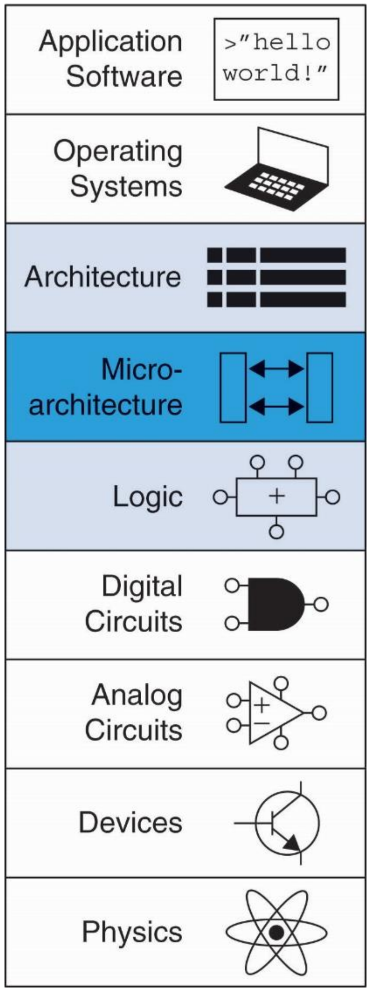
31:28	27:26	25	24	23	22	21	20	19:8	7:0
cond	op	X	X	U	X	X	X	X	imm8
4 bits	2 bits	funct 6 bits						12 bits	8 bits

Branch
```
B{cond} LABEL
LABEL encoded as
           #±imm8
```

Take a printout of this page and keep it for quick reference for the rest of this chapter

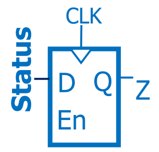
Microarchitecture

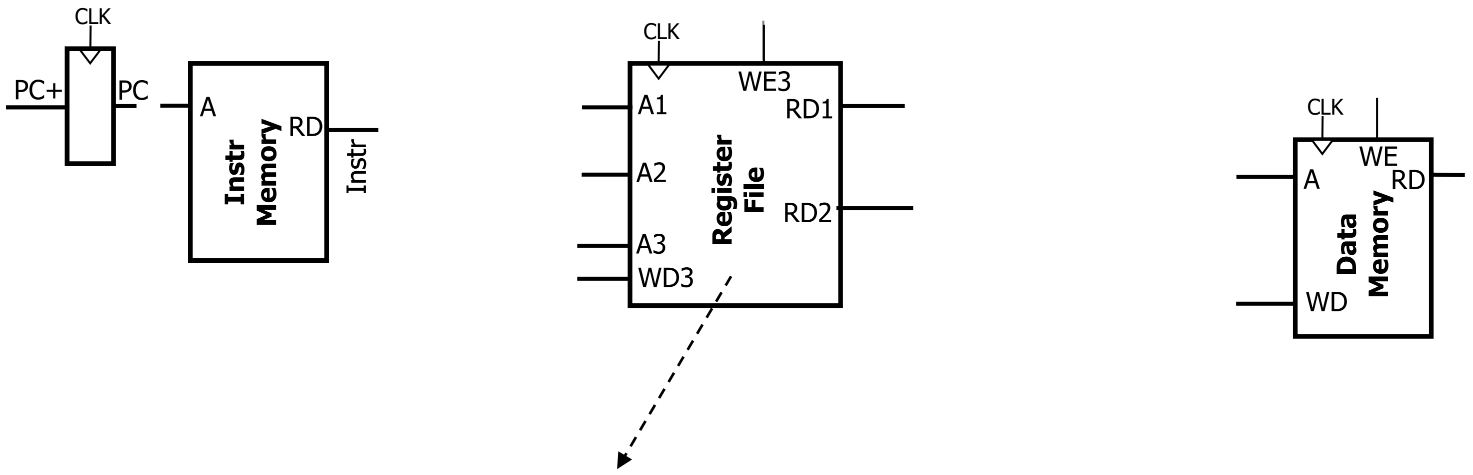
- Processor
  - Datapath: functional blocks
  - Control: control signals
- Basic styles
  - **Single-cycle**: Each instruction executes in a single cycle
  - **Multicycle**: Each instruction is broken up into series of shorter steps
  - **Pipelined**: Each instruction broken up into series of steps & multiple instructions execute at once



Architectural State Elements

- Architectural state determines everything about a processor (state of program execution)
  - 16 registers (including PC)
  - Memory
  - Status register (flags)
  - Changed by all instructions, at the clock edge following it

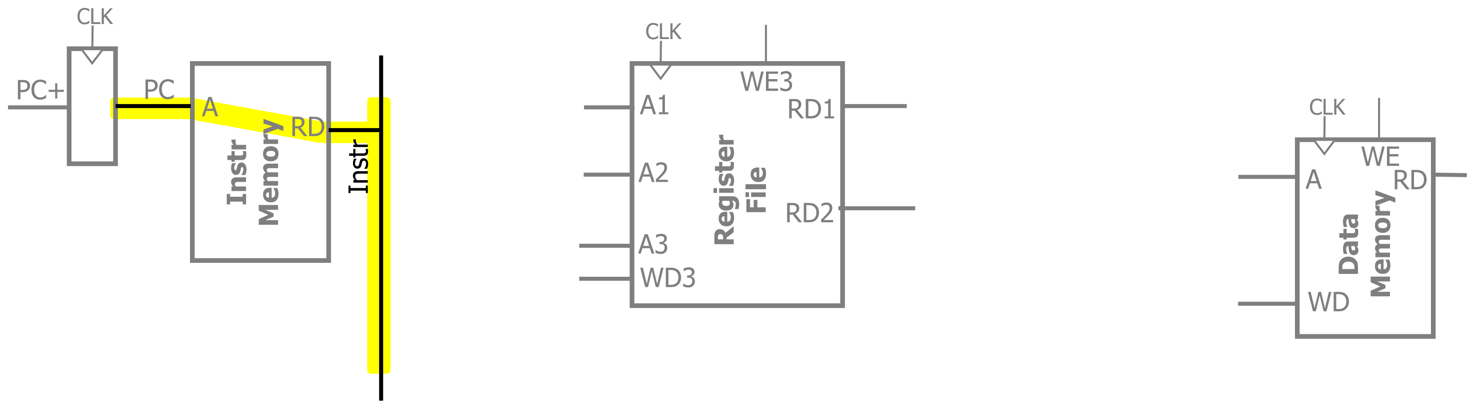




15 registers (i.e., all except PC) shown as a register file

Datapath : LDR – Fetch

- STEP 1: Fetch instruction
- Assume memories can be read combinationally – no clock

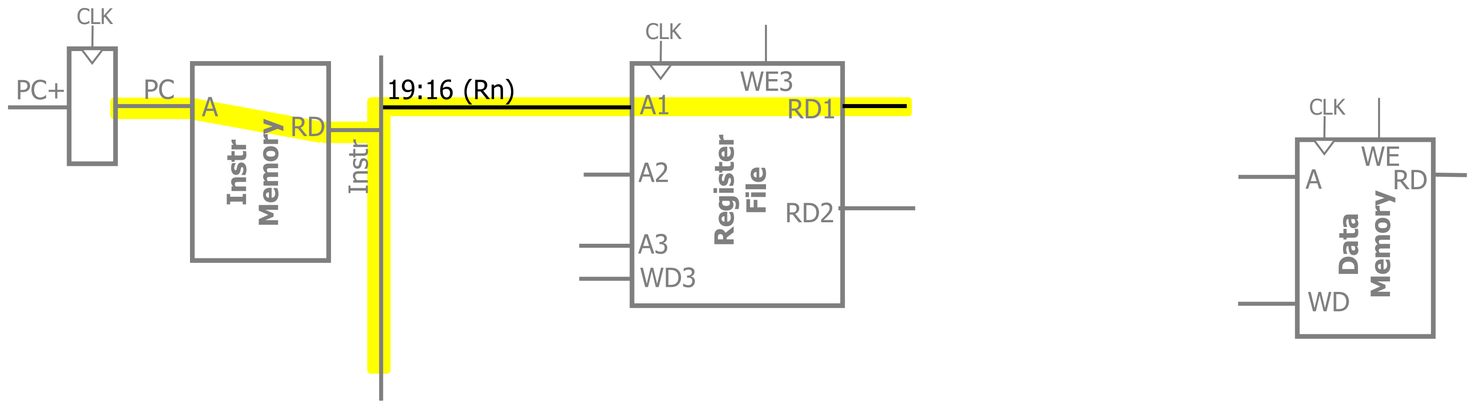


Datapath : LDR – Read RF

31:28	27:26	25	24	23	22	21	20	19:16	15:12	11:8	7:0
X	op	X	P	U	X	W	L	Rn	Rd	X	imm8
4 bits	2 bits	funct 6 bits						4 bits	4 bits	4 bits	8 bits

```
LDR Rd, [Rn, #±imm8]
```

- STEP 2: Read first source operand (*Rn*) from Register File
- Reading register doesn't need clock

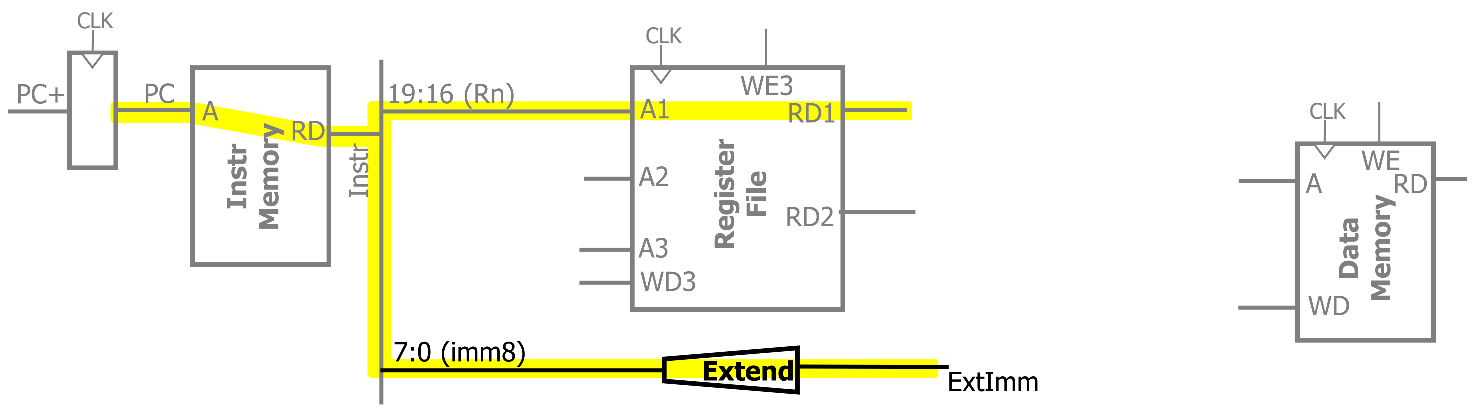


Datapath : LDR – Extend Immediate

31:28	27:26	25	24	23	22	21	20	19:16	15:12	11:8	7:0
X	op	X	P	U	X	W	L	Rn	Rd	X	imm8
4 bits	2 bits	funct 6 bits						4 bits	4 bits	4 bits	8 bits

```
LDR Rd, [Rn, #±imm8]
```

- STEP 3: Extend the immediate to make it 32 bit
  - Insert 24 zeros in front (24 MSBs hard-wired to 0s)

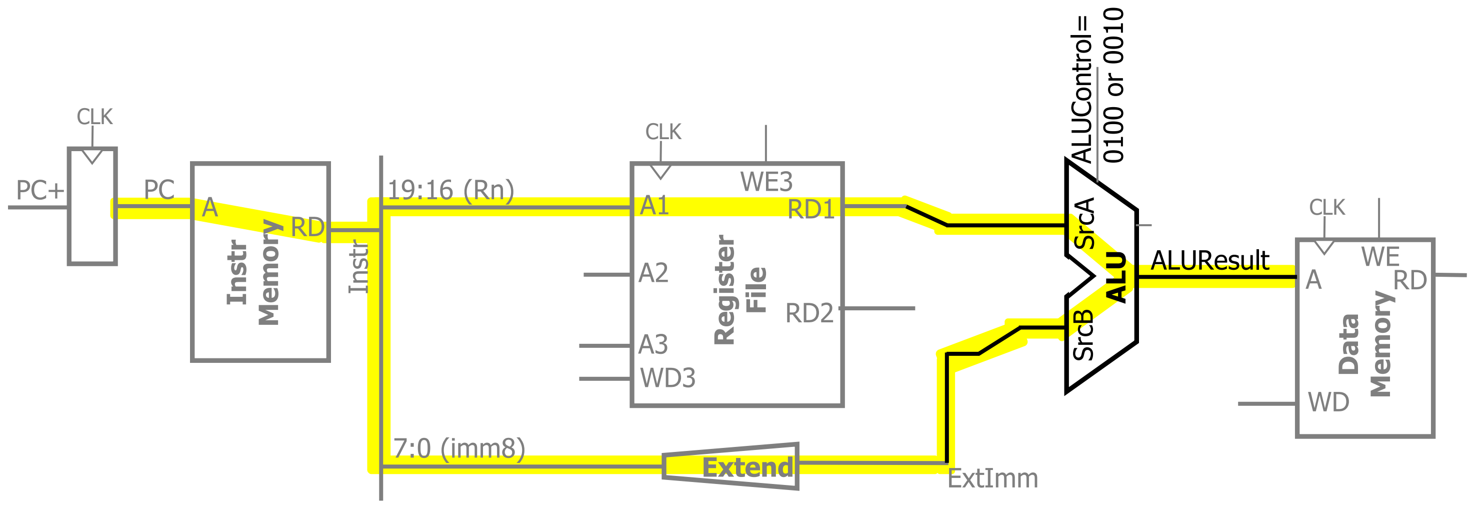


Datapath : LDR – Data Mem Address

31:28	27:26	25	24	23	22	21	20	19:16	15:12	11:8	7:0
X	op	X	P	U	X	W	L	Rn	Rd	X	imm8
4 bits	2 bits	funct 6 bits						4 bits	4 bits	4 bits	8 bits

```
LDR Rd, [Rn, #±imm8]
```

- STEP 4: Compute the data memory address
- ALU performs add or sub depending on the offset sign

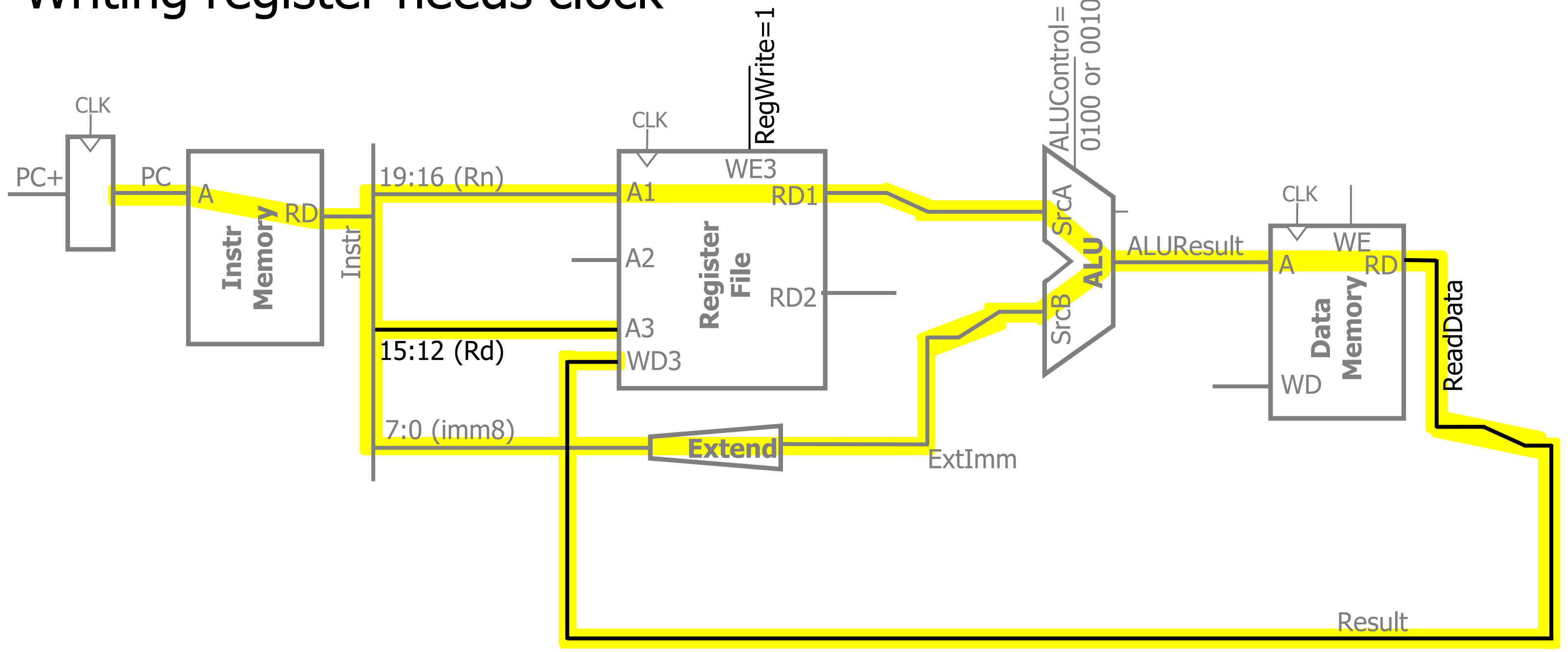


Datapath : LDR – Read Data Mem

31:28	27:26	25	24	23	22	21	20	19:16	15:12	11:8	7:0
X	op	X	P	U	X	W	L	Rn	Rd	X	imm8
4 bits	2 bits	funct 6 bits						4 bits	4 bits	4 bits	8 bits

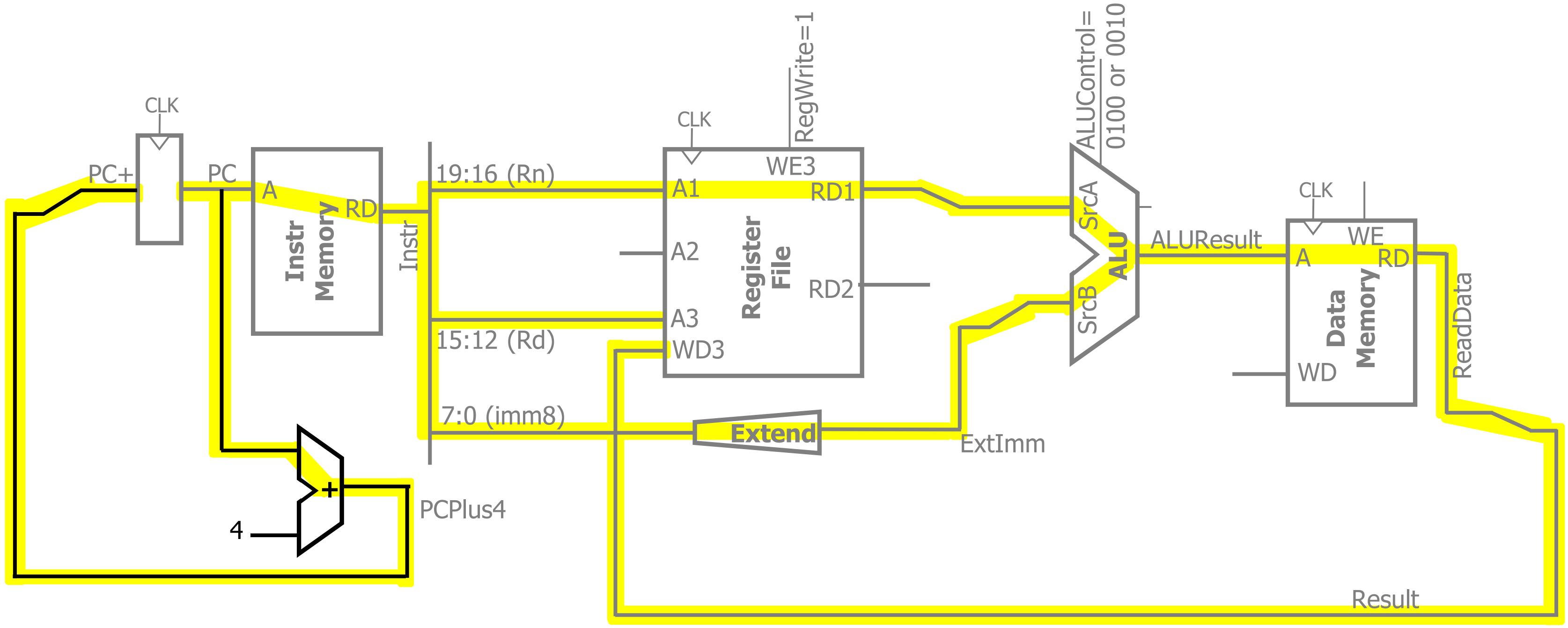
```
LDR Rd, [Rn, #±imm8]
```

- STEP 5: Read data from memory and write it back to register file
  - Writing register needs clock



Datapath : LDR – PC Increment

- STEP 6: Determine address of next instruction
- LDR Done!

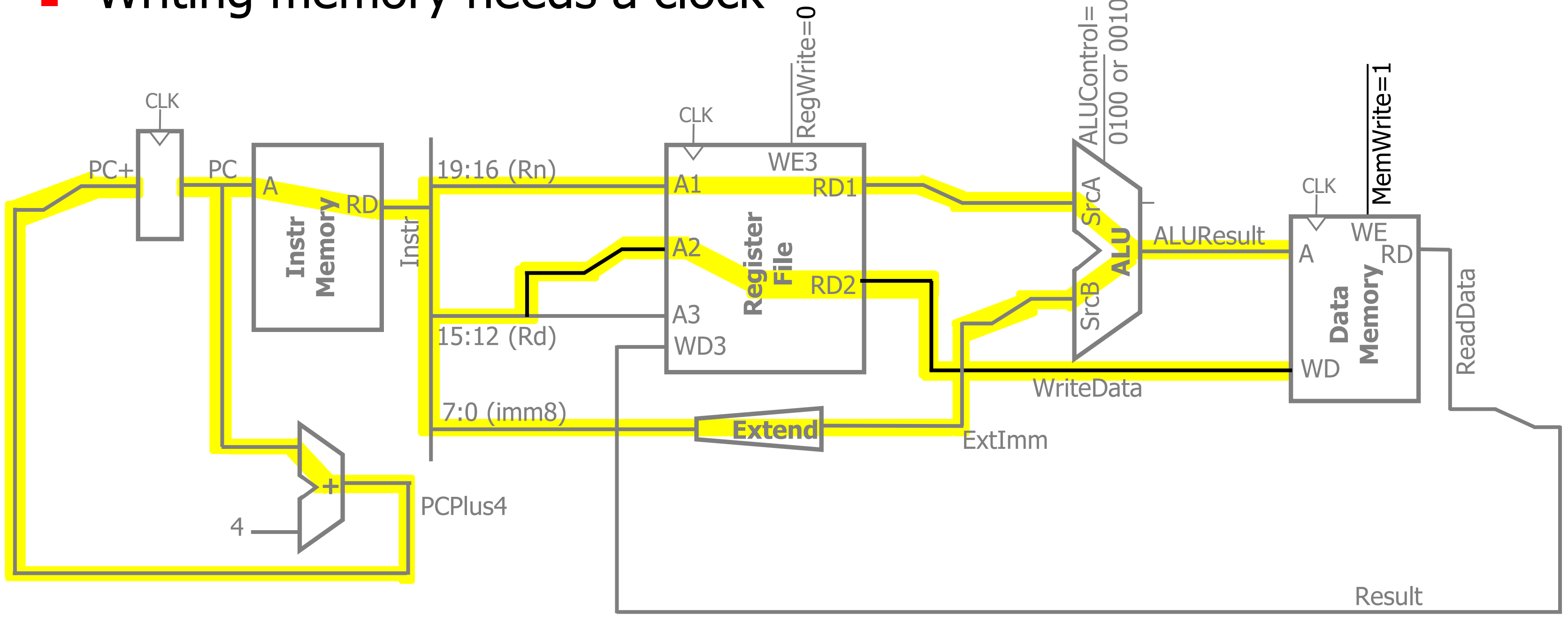


Datapath : STR

31:28	27:26	25	24	23	22	21	20	19:16	15:12	11:8	7:0
X	op	X	P	U	X	W	L	Rn	Rd	X	imm8
4 bits	2 bits	funct 6 bits						4 bits	4 bits	4 bits	8 bits

```
STR Rd, [Rn, #±imm8]
```

- Write data in *Rd* to memory
  - Note that *Rd* is a source operand for STR
  - Writing memory needs a clock



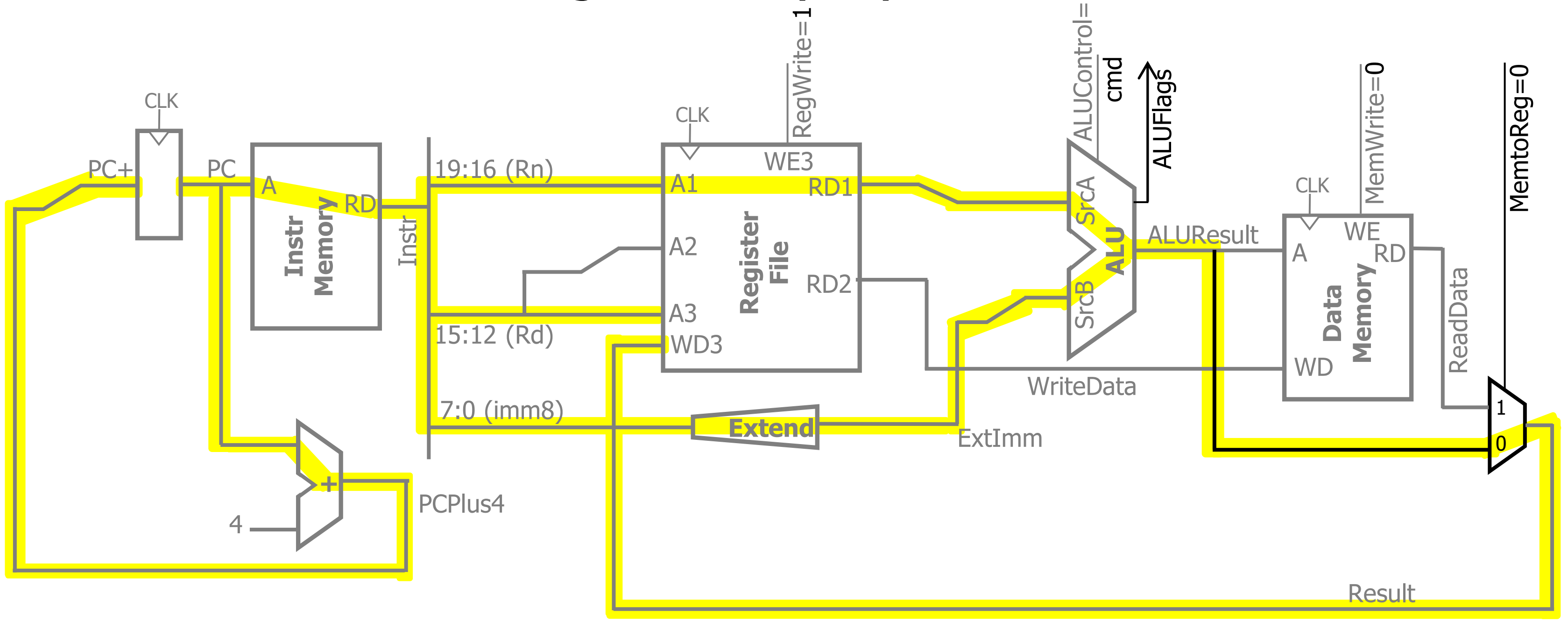
Datapath : Data Processing (Immediate)

31:28	27:26	25	24:21	20	19:16	15:12	11:8	7:0
X	op	I	cmd	S	Rn	Rd	X	imm8
4 bits	2 bits	funct 6 bits			4 bits	4 bits	4 bits	8 bits

DP immediate

```
OP{S} Rd, Rn, #imm8
OP => ADD, AND,..
```

- With immediate *Operand2*
  - Read from *Rn* and *ExtImm*
  - Write ALUResult to register file (*Rd*)



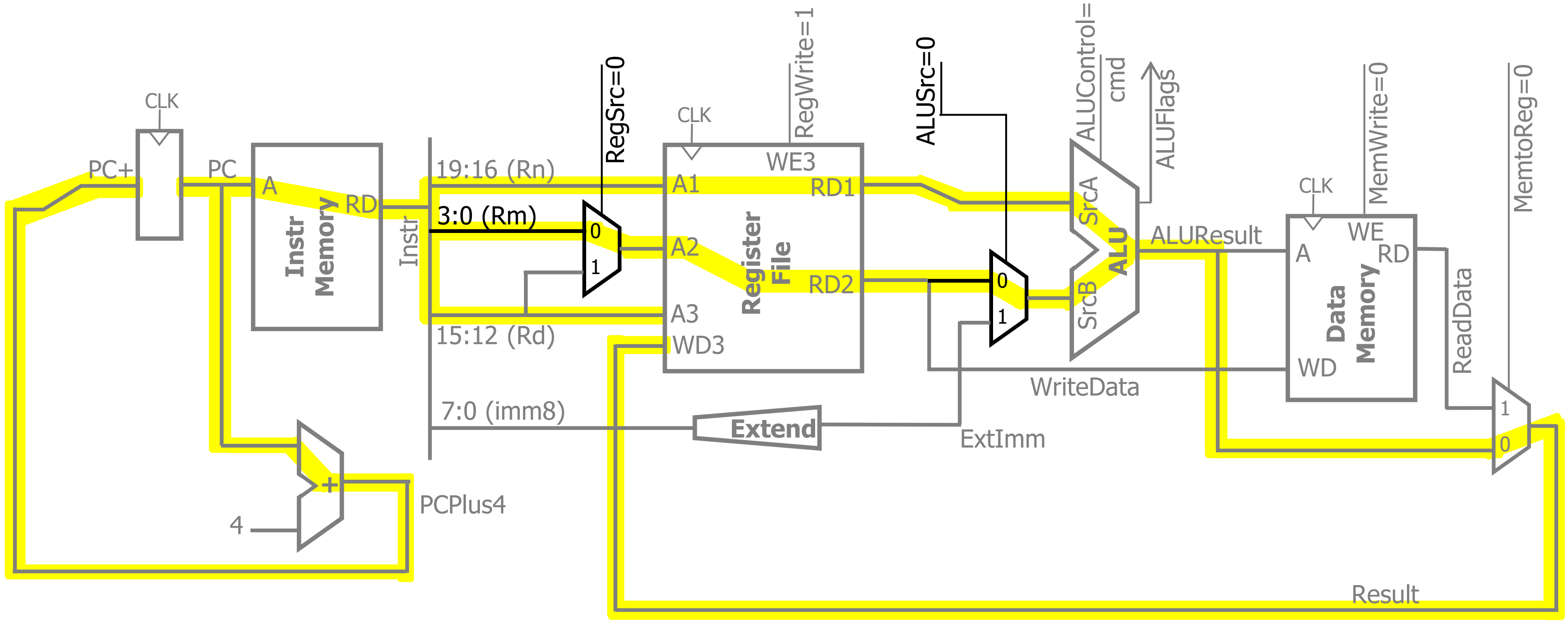
Datapath : Data Processing (Register)

31:28	27:26	25	24:21	20	19:16	15:12	11:5	4	3:0
X	op	I	cmd	S	Rn	Rd	X	M	Rm
4 bits	2 bits	funct 6 bits			4 bits	4 bits	7 bits	1 bit	4 bits

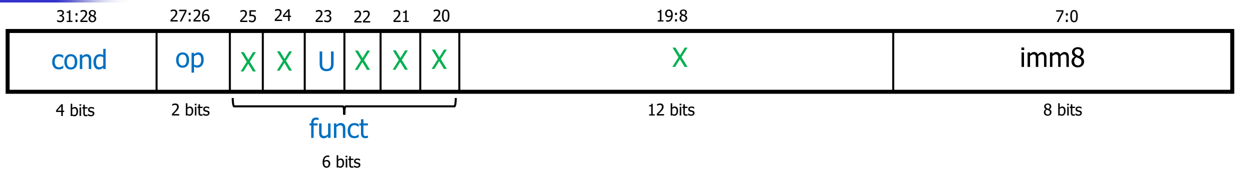
DP register

```
OP{S} Rd, Rn, Rm
OP => ADD, AND,..
```

- With register *Operand2*
  - Read from *Rn* and *Rm* (instead of *ExtImm*)

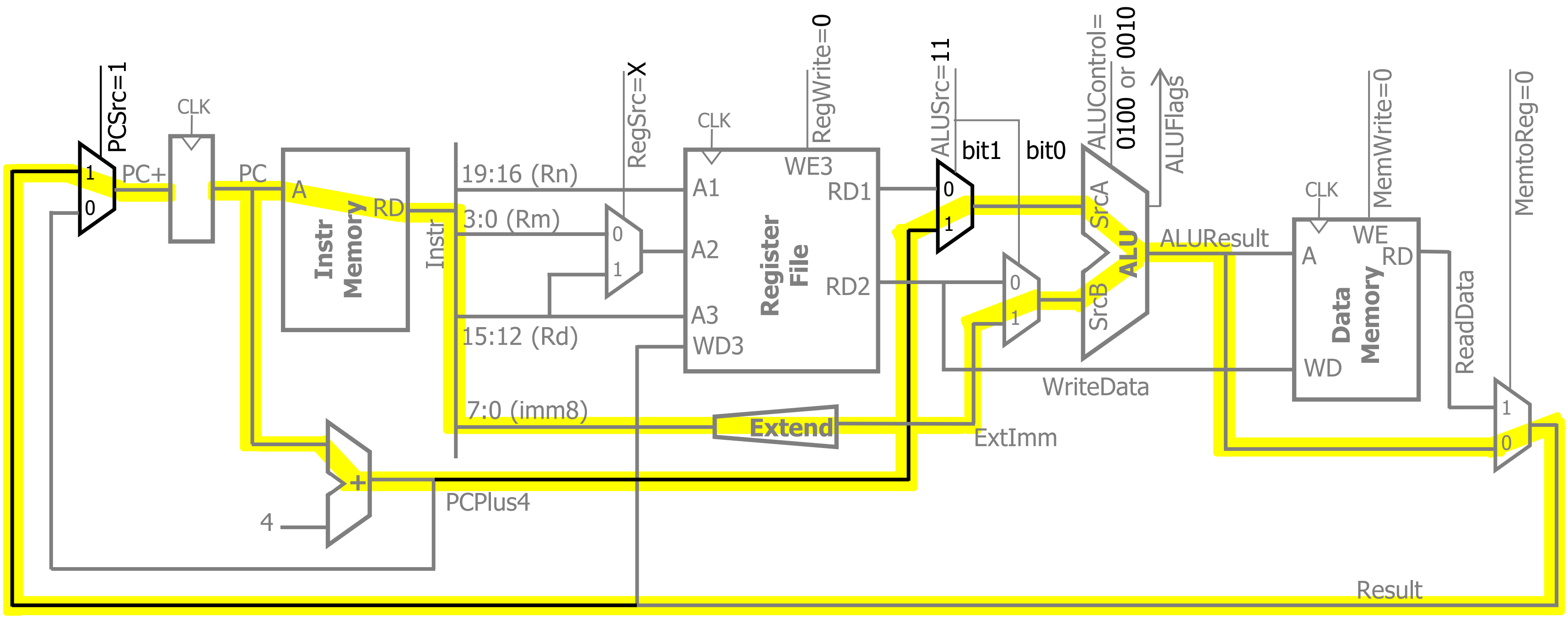


Datapath : Branch

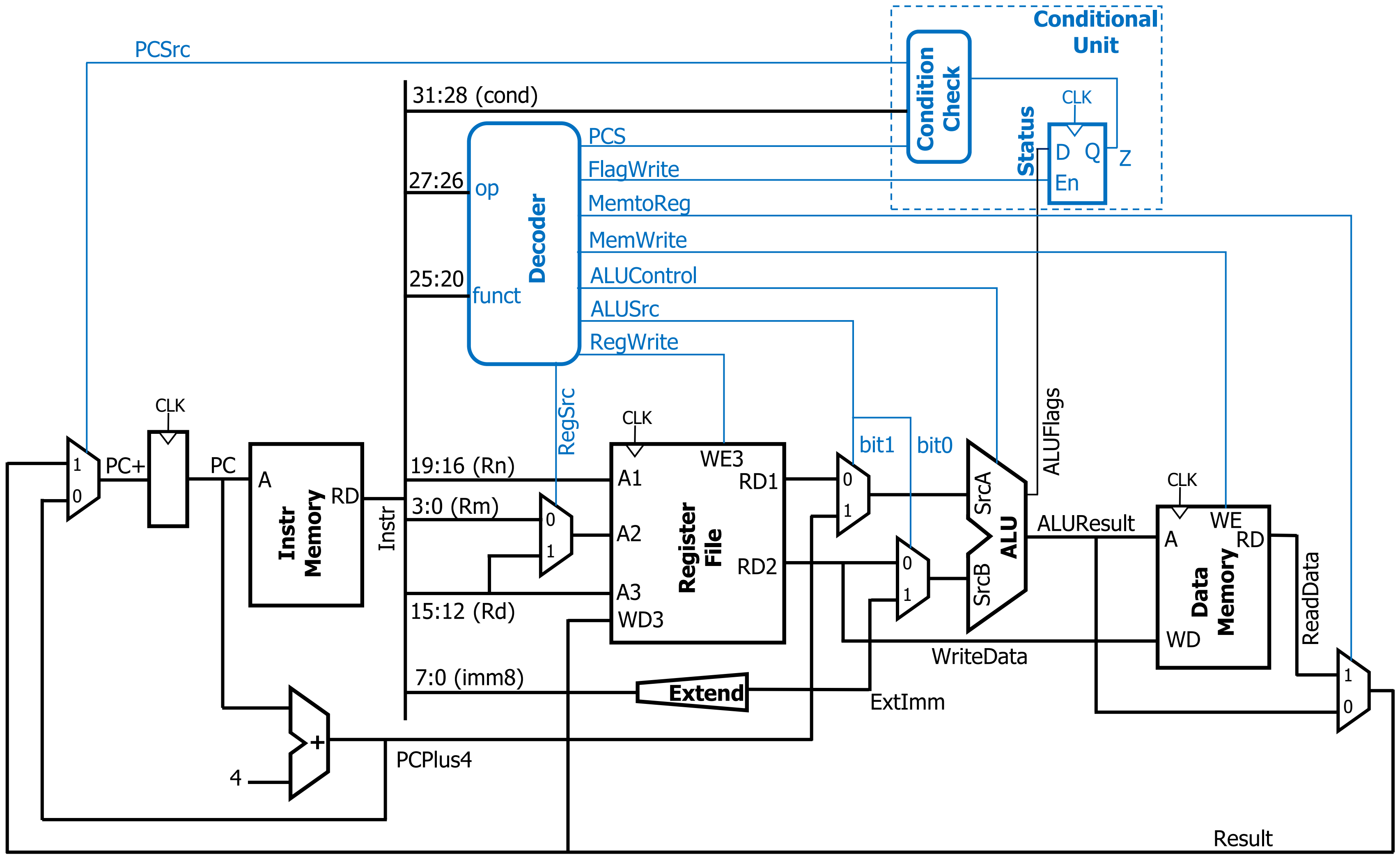


Branch
`B{cond} LABEL`
`LABEL encoded as #±imm8`

- Calculate branch target address
  - BTA = (PC + 4) +/- (*ExtImm*)



Single-Cycle Processor with Control



Control Unit Design

All expressions use C syntax; all values are in binary (0b prefix not explicitly written for convenience)

Decoder

- PCS = (op==10)
  - op = Instr[27:26]
  - Asserted only for branch, to write branch target to PC. Passed through conditional unit before being used in the datapath
- FlagWrite = (op==00) && (S==1)
  - S = funct[0] = Instr[20]
  - Asserted for DP with S suffix, as only they modify flags
- MemtoReg = (op==01) && (L==1)
  - L = funct[0] = Instr[20]
  - Asserted only for load, as the destination register gets data read from the data memory (what should be MemtoReg for store?)
- MemWrite = (op==01) && (L==0)
  - Asserted only for store, as store alone writes to the data memory

Control Unit Design

- ALUControl = (op==00)? cmd : (U? 0100:0010)
  - U = funct[3] = Instr[23]
  - 0100 – ALUControl for addition, 0010 – ALUControl for subtraction
  - For DP, ALUControl is cmd. For memory and branch, U bit decides whether imm8 is added or subtracted (i.e., whether the offset is positive or negative)
- ALUSrc[0] = !((op==00) && (I==0))
  - I = funct[5] = Instr[25]
  - For all except DP with register as Operand2, ALU_SrcB is immediate
- ALUSrc[1] = PCS
  - ALU_SrcA is PCPlus4 only for branch (doesn't matter whether branch is taken or not. ALUResult is discarded when the branch is not taken anyway)

Control Unit Design

- RegWrite = (op==00) || ((op==01) && (L==1))
  - All DP instructions and load write to a destination register, branch and store doesn't
- RegSrc = MemWrite
  - For store, RA2 = Rd. For all other instructions reading a second register (i.e., DP Reg), RA2 is Rm

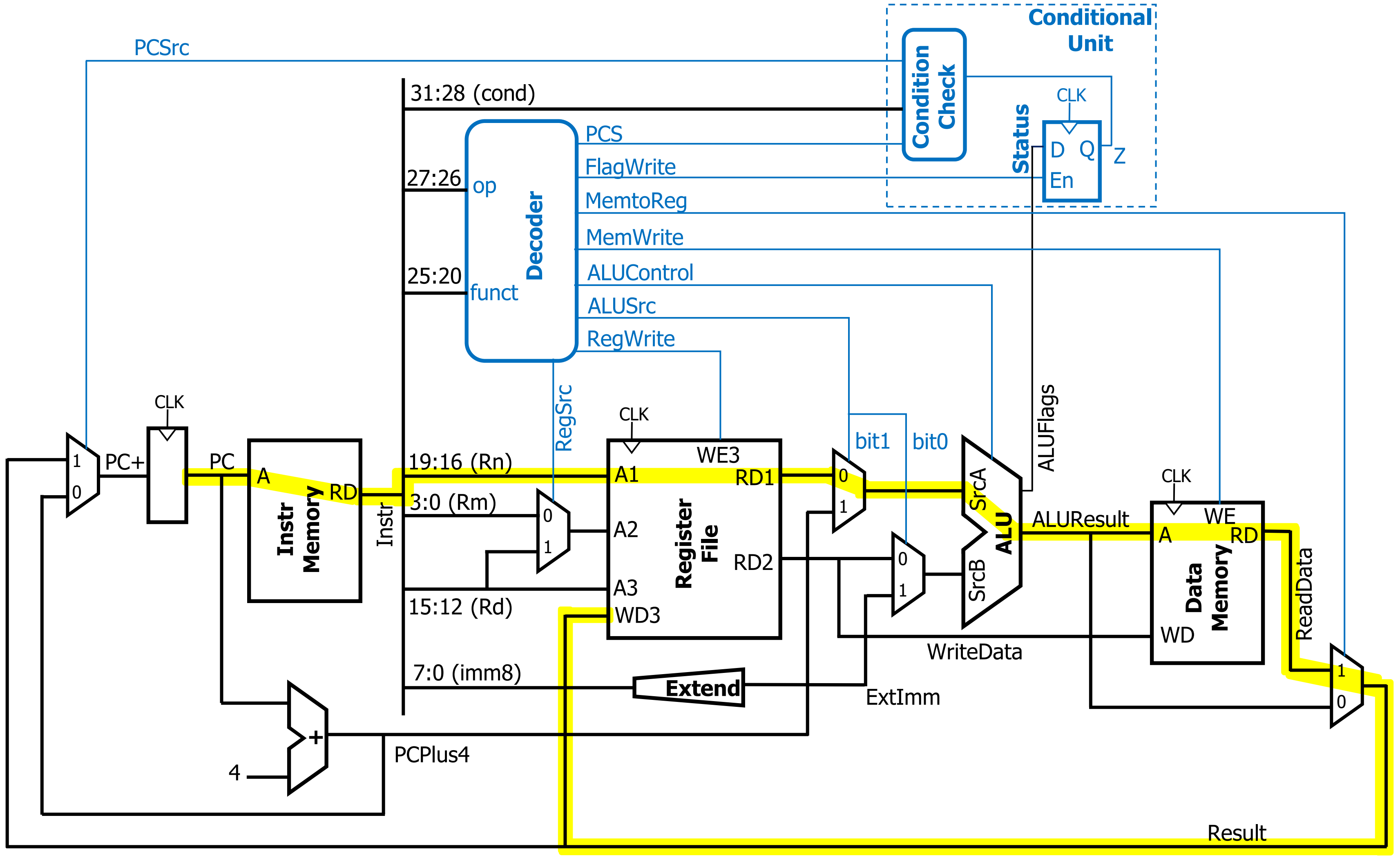
Condition Check

- PCSrc = PCS && ((cond==0000) ? (Z==1) : 1)
  - For a branch instruction
    - When the condition specified is EQ (0000) and when Z flag is set, branch is taken
    - When the condition specified is AL (1110), branch is taken irrespective of the flags. For simplicity, we just ignore flags if the condition specified is not EQ
  - This will cause ALUResult (PCPlus4+/-imm8) to be written to PC instead of PCPlus4

Extended Functionality : CMP

- Recall : CMP does subtraction, sets the flags and discards the result
  - Has S = 1 (even though there is no explicit suffix S for CMP)
  - cmd = 1010 (different from SUBS)
  - Similar to SUBS, but the result is not written to a register. So we need to modify RegWrite signal
  - *Rd* is irrelevant (*Rd* is designated as SBZ = should be 0s in actual ARM instruction set)
- RegWrite = ((op==00) && !NoWrite) || ((op==01) && (L==1))
  - Where NoWrite = (cmd==1010)
  - NoWrite can be easily extended to accommodate other DP instructions (TST, TEQ, CMN) which do not write the result
- No change to datapath needed for implementation!

Single-Cycle Performance



- T_C (and hence, performance) limited by critical path (LDR)

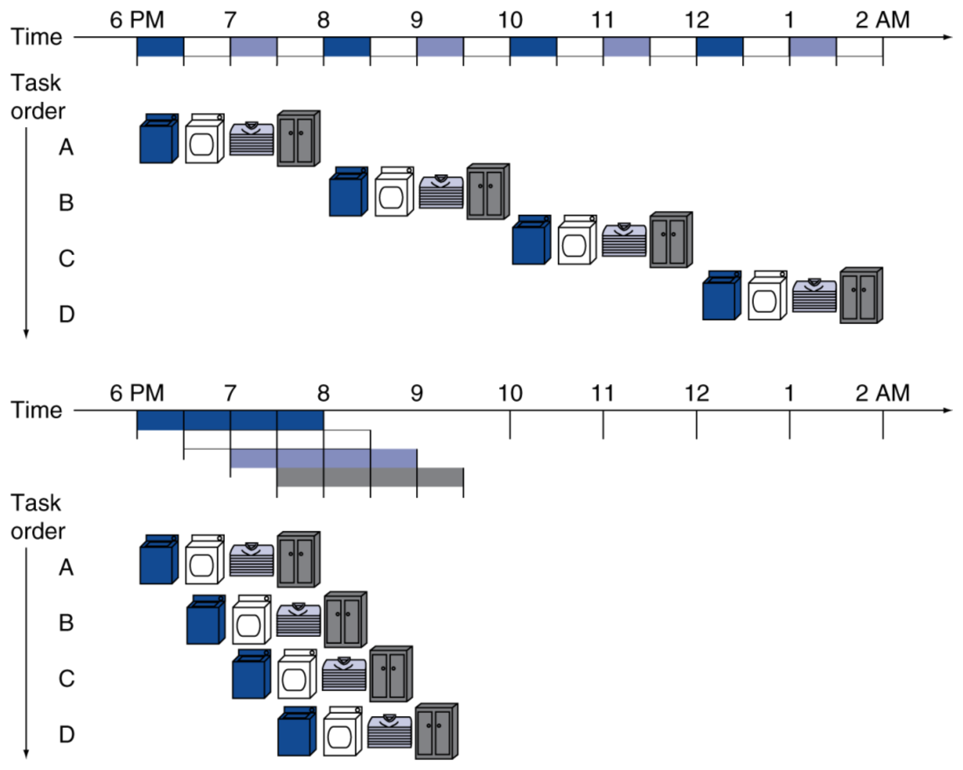
Single Cycle Design Summary

- Single-cycle - fetch, decode and execute each instruction in one clock cycle
  - (+) simple
  - (–) no datapath resource can be used more than once per instruction, so some must be duplicated
    - separate memories for instruction and data
    - 2 adders/ALUs
  - (–) cycle time limited by longest instruction (LDR)

- How Can We Make It Faster?
  - Pipelining
  - Superscalar
  - …… stay tuned for Chapter 6!

Pipelining Analogy. Coming Soon!

- Pipelined laundry: overlapping execution
  - Parallelism improves performance



- Four loads:
  - Speedup = 8/3.5 = 2.3
- Non-stop:
  - Speedup = 2n/(0.5n + 1.5) ≈ 4 = # of stages

Other Formats : Self Reading

Note :

- No need to memorize. Required info will be given in the exam if need be
- Some bits which were left as don’t cares are used in the following slides

DP Operations : cmd

cmd	Instr	Operation	Notes
0000	AND	Logical AND	
0001	EOR	Logical Exclusive OR	
0010	SUB	Subtract	
0011	RSB	Reverse Subtract	
0100	ADD	Add	
0101	ADC	Add with Carry	
0110	SBC	Subtract with Carry	
0111	RSC	Reverse Subtract with Carry	
1000	TST	Test Update flags after AND	S=1. Rd is not used. Encode as 0b0000
1001	TEQ	Test Equivalence Update flags after EOR	S=1. Rd is not used. Encode as 0b0000
1010	CMP	Compare Update flags after SUB	S=1. Rd is not used. Encode as 0b0000
1011	CMN	Compare Negated Update flags after ADD	S=1. Rd is not used. Encode as 0b0000
1100	ORR	Logical OR	
1101	MOV	Move	Rn is not used. Encode as 0b0000
1110	BIC	Bit Clear	
1111	MVN	Move Not	Rn is not used. Encode as 0b0000

Note : Multiplication is not one of the 16 ALU operations, though it is considered a DP operation. Multiplication is done in a separate multiplication unit and is a bit different from other DP operations.

Multiply Instruction Format

31:28	27:26	25	24:21	20	19:16	15:12	11:8	7:5	4	3:0
X	op	I	cmd	S	Rn	Rd	Rs	X	M	Rm
4 bits	2 bits		funct 6 bits		4 bits	4 bits	4 bits	3 bits	1 bit	4 bits

- `MUL Rd, Rm, Rs` (*Rd* = *Rm** *Rs*)
- `MLA Rd, Rm, Rs, Rn` (*Rd* = *Rn* + *Rm** *Rs*)
- cmd = 0b0000 for MUL, 0b0001 for MLA
- M = 0b0 -> usual DP instructions such as ADD, AND,..

  0b1 -> MUL and MLA
- MUL does not use *Rn*
- Assume MUL and MLA does not set any flags (S bit is 0b0) and cannot take immediate operands (I bit is 0b0)
- Note that M bit is checked only when I bit is zero. When I bit is 1, bit 7:0 is interpreted as immediate and there is no M bit
- Design Hint : You will need a register file with 3 read ports to implement MLA, as 3 registers need to be read simultaneously

Memory Instruction Format

31:28	27:26	25	24	23	22	21	20	19:16	15:12	11:8	7:0
X	op	X	P	U	X	W	L	Rn	Rd	X	imm8
4 bits	2 bits	funct 6 bits						4 bits	4 bits	4 bits	8 bits

- funct
  - U : Add
    - 0b1 -> positive offset; 0b0 -> negative offset
  - L : Load
    - 0b1 -> load; 0b0 -> store
  - P : Preindex
  - W : Writeback
  - PW = 0b00 -> postindex 0b01 -> unsupported
    0b10 -> offset 0b11 -> preindex

Note: PC relative mode is identical to offset mode, i.e., PW=0b10, with *Rn*=R15 and offset computed automatically by the assembler.

Note that R15 is always read as PC+4. However, the processor we designed does not support reads from R15 except for branch instruction.

Branch Condition Codes (cond)

- Flags are set by instructions with suffix S
  - Example : ADDS affects flags, ADD doesn't
  - Exceptions : CMP, CMN, TST, TEQ which are used only to set flags (result is discarded)

cond	Mnemonic	Name	Condition Checked
0000	EQ	Equal	Z
0001	NE	Not equal	$\bar{Z}$
0010	CS / HS	Carry set / Unsigned higher or same	C
0011	CC / LO	Carry clear / Unsigned lower	$\bar{C}$
0100	MI	Minus / Negative	N
0101	PL	Plus / Positive of zero	$\bar{N}$
0110	VS	Overflow / Overflow set	V
0111	VC	No overflow / Overflow clear	$\bar{V}$
1000	HI	Unsigned higher	$\bar{Z}C$
1001	LS	Unsigned lower or same	$Z\ OR\ \bar{C}$
1010	GE	Signed greater than or equal	$\overline{N \oplus V}$
1011	LT	Signed less than	$N \oplus V$
1100	GT	Signed greater than	$\bar{Z}(\overline{N \oplus V})$
1101	LE	Signed less than or equal	$Z\ OR\ (N \oplus V)$
1110	AL (or none)	Always / unconditional	ignored

Interpretation based on SUBS/CMP