CSM 61C Single-Cycle Datapath and Control
Fall 2020

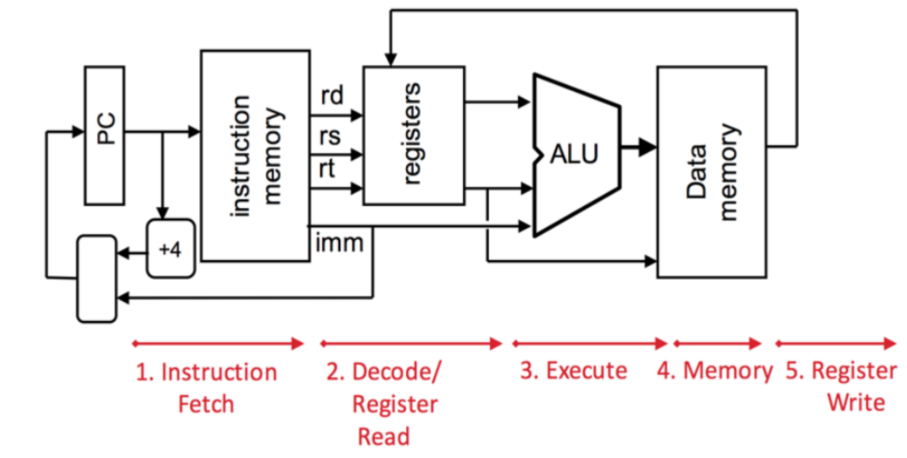
Mentoring 6: March 16, 2020

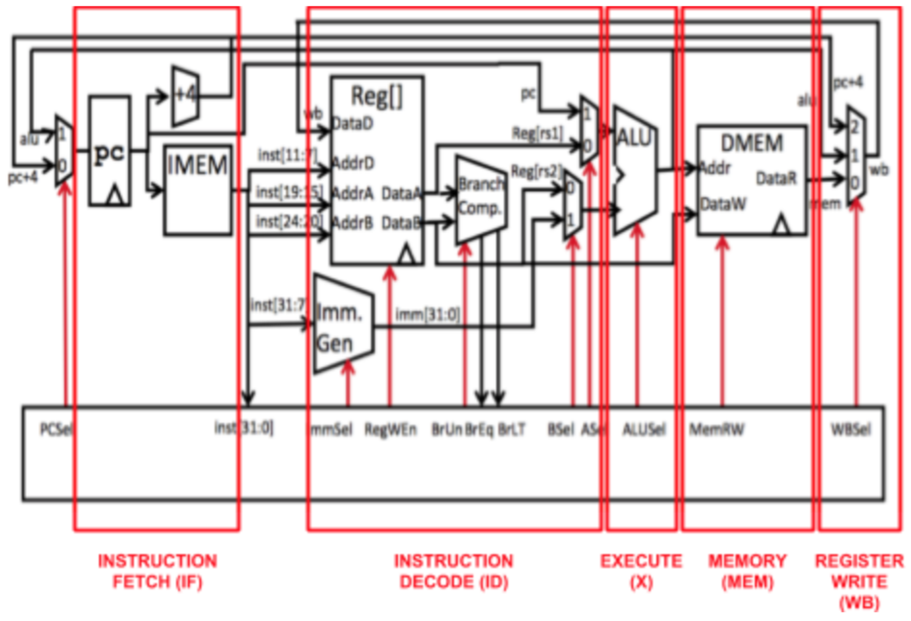
1 Single-Cycle Datapath and Control

1.1 **5 Stages of a Single Cycle CPU:**

- Instruction Fetch (IF) - Fetch from memory (IMEM)
- Instruction Decode (ID) - Decode instruction
- Execute (EX) - Execute operation (arithmetic, shifting, etc) using ALU
- Memory Access (MEM) - Load and store instructions access memory
- Write Back to Register (WB) - Write instruction back to RegFile

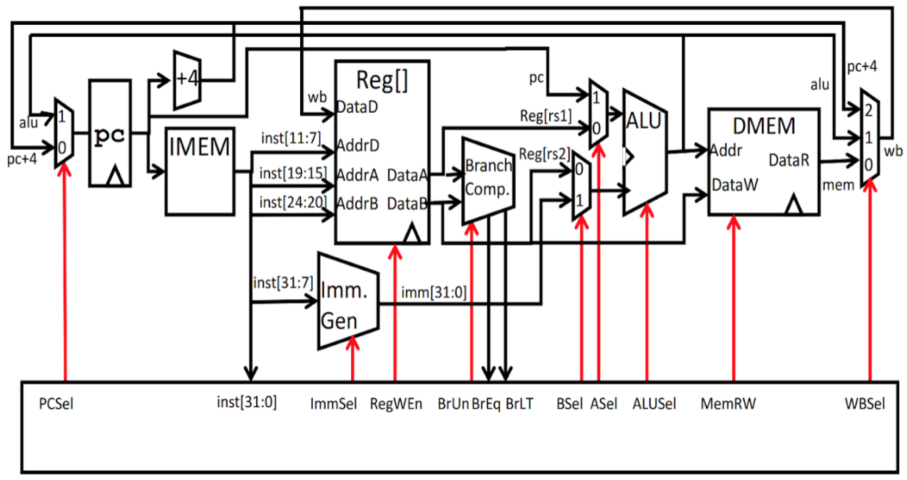
Datapaths: A Visual Approach





1.2 Control Logic

A controller send signals to our circuit, telling which pieces to perform what operations. Not all control signals matter for every instruction: for example, R-type instructions ignores the output from the immediate generator. Control signals are used to pick between mux inputs in order to perform the correct operation. They are embedded within the actual machine code for an instruction.



Control Inputs

Signal:	**inst[31:0]**	**BrEq**	**BrLT**
Purpose:	Sends the current instruction to control	(DataA == DataB) ? 1 : 0	(DataA < DataB) ? 1 : 0

Control Outputs

Signal:	**Purpose:**
PCSel	Next instruction location
ALUSel	What operation to perform.
RegWEn	Do we allow the register value to be updated by enabling writes?
ImmSel	Format the immediate properly.
MemRW	Read or write to mem.
WBSel	What value to write back.
BrUn	Branch signed or unsigned.
ASel/BSel	Pick between the inputs for ALU.

2 Game of Signals

2.1 Fill out the control signals for the following instructions (put an X if the signal does not matter). For ImmSel, write the corresponding instruction type.

Instr	BrEq	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
add	X	0	X	X	0	0	Add	0	1	1
lw										
bge										
sw										
auipc										
jal										

2.2 We want to expand our instruction set from the base RISC-V ISA (RV32I) to support some new instructions. You can find the canonical single-cycle datapath above. For the proposed instruction below, choose ONE of the options below.

1. Can be implemented without changing datapath wiring, only changes in control signals are needed. (i.e. change existing control signals to recognize the new instruction)
2. Can be implemented, but needs changes in datapath wiring, only additional wiring, logical gates and muxes are needed.
3. Can be implemented, but needs change in datapath wiring, and additional arithmetic units are needed (e.g. comparators, adders, shifters etc.).
4. Cannot be implemented.

(Note that the options from 1 to 3 gradually add complexity; thus, selecting 2 implies that 1 is not sufficient. You should select the option that changes the datapath the least (e.g. do not select 3 if 2 is sufficient). You can assume that necessary changes in the control signals will be made if the datapath wiring is changed.)

(a) Allowing software to deal with 2's complement is very prone to error. Instead, we want to implement the negate instruction, `neg rd rs1`, which puts `-R[rs1]` in `R[rd]`.

(b) Sometimes, it is necessary to allow a program to self-destruct. Implement `segfault rs2, offset(rs1)`. This instruction compares the value in `R[rs2]` and the value in `MEM[R[rs1]+offset]`. If the two values are equal, write `0` into the `PC`; otherwise treat this instruction as a `NOP`.

3 Single Cycle CPU Timing Practice

The delays of a circuit elements are given as follows:

Clk-to-Q	RegFile Read	Mux	ALU	MEM Read
5ns	20ns	5ns	100ns	150ns

MEM Write	Branch Comp	Imm Gen	RegFile Setup
200ns	50ns	25ns	4ns

3.1 Ignoring the length of a clock cycle, how long does it take to execute the instruction:

(a) `lui t0, 0x1234`

(b) `jal ra, 0b1100`

(c) `beq x0, x5, 0b1100`

3.2 What is the length of the critical path in the CPU? Which instruction exercises the critical path? Highlight its path on the diagram above starting from the PC.

3.3 What is the fastest possible clock for this datapath without any violations?

4 movz and movnz

Consider adding the following instruction to RISC-V (disregard any existing/similar definition on the green sheet):

Instruction	**Operation**
`movz rd, rs1, rs2`	`if (R[rs1] == 0) R[rd]<-R[rs2]`
`movnz rd, rs1, rs2`	`if (R[rs1] != 0) R[rd]<-R[rs2]`

4.1 Translate the following C code using movz and movnz. Do not use branches.

C Code

```
// a -> s0, b-> s1, c-> s2
int a = b < c ? b : c;
```

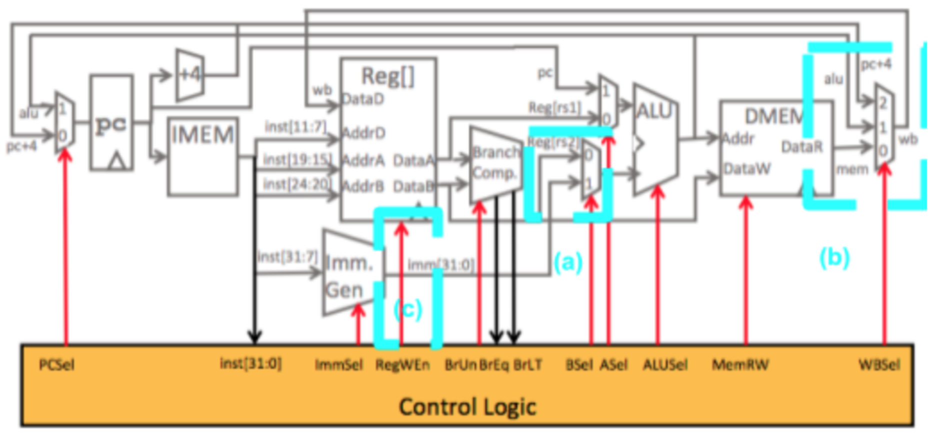
RISC-V Code

```
slt t0, s1, s2
```

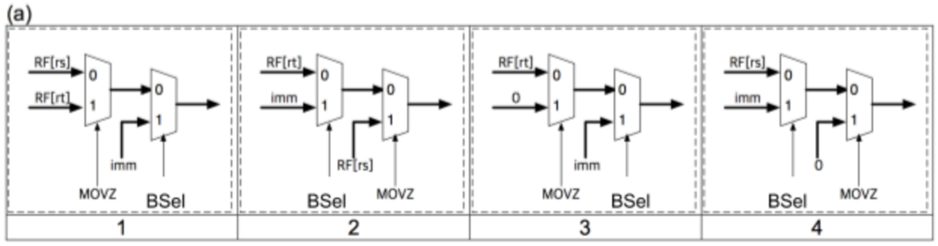
____

____

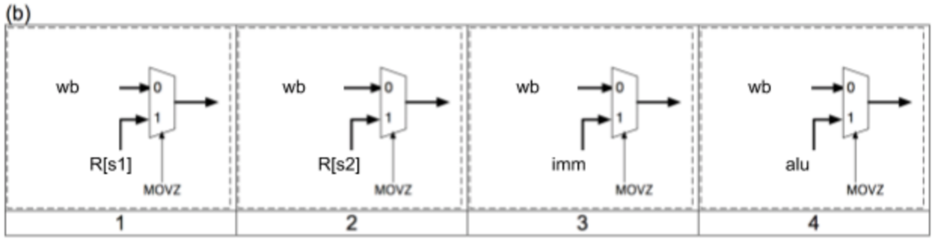
4.2 Implement If movz (but not movnz) in the datapath. Choose the correct implementation for (a), (b), and (c). Note that you do not need to use all the signals provided to each box, and the control signal MOVZ is 1 if and only if the instruction is **movz**. In the following diagrams, `RF[rs]` refers to `R[rs1]` and `RF[rt]` refers to `R[rs2]`. `ALUZero = 1` if the `A` input of the `ALU` is zero.



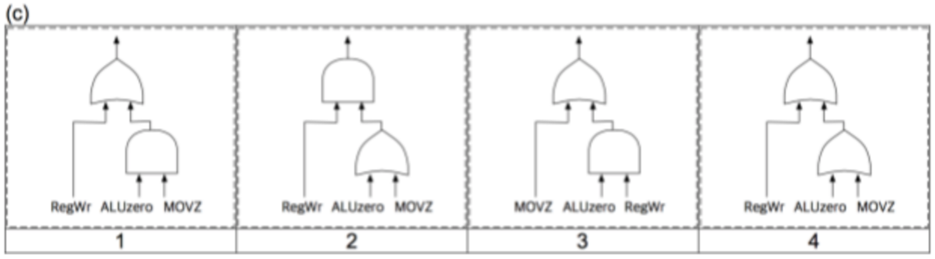
(a)



(b)



(c)



4.3 Generate the control signals for movz. The values should be 0, 1, or X (don't care) terms. You must use don't care terms where possible.

MOVZ	PCSel	ImmSel	RegWEn	BrUn	BSel	ASel	ALUSel	MemRW	WBSel
1									