ECE154A — Discussion 07

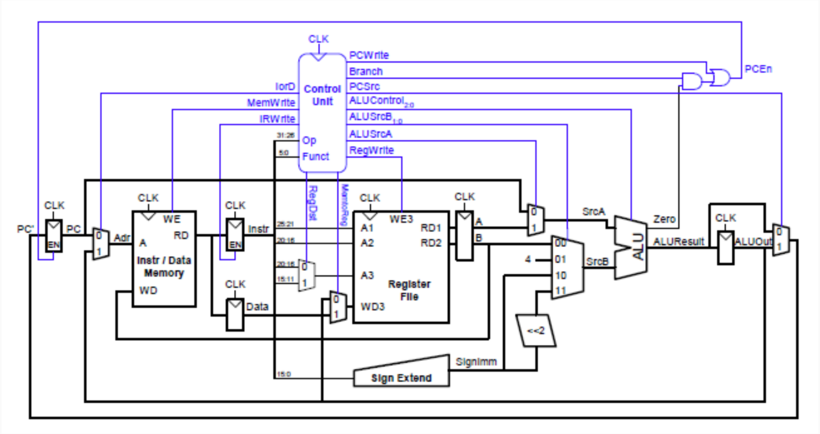
George Higgins Hutchinson

November 12, 2021

Keep your eyes open for...

- PSet 4: due Friday, November 12
- Lab 4: due Monday, November 15
- Lat 5: assigned soon

MIPS multi-cycle



Why multi-cycle?

- Smaller? Cheaper?
- Faster?
- Simpler to implement?
- Flexible (how hard to add complicated instructions?)

Why multi-cycle?

- Smaller? Cheaper?
  Probably! Need new registers, more complicated (stateful) control, but only one memory unit and some extra computation folded into ALU.
- Faster?
  Probably not! Same "critical paths" for a full instruction, but now there's extra registers (setup and clk-2-q delay)
- Simpler to implement?
  Well, you decide this one after lab 5. Historically, early intel cores were multicycle with a bus and microcode.
- Flexible (how hard to add complicated instructions?)
  Probably easier, since you don't have a single time constraint to meet anymore.

H&H 7.10

A fabrication issue has caused one control signal in your processor to be hard-wired to V_{DD}. Which instructions are now broken?

a) MemtoReg

b) ALUOp[0]

c) PCSrc

Custom Instructions

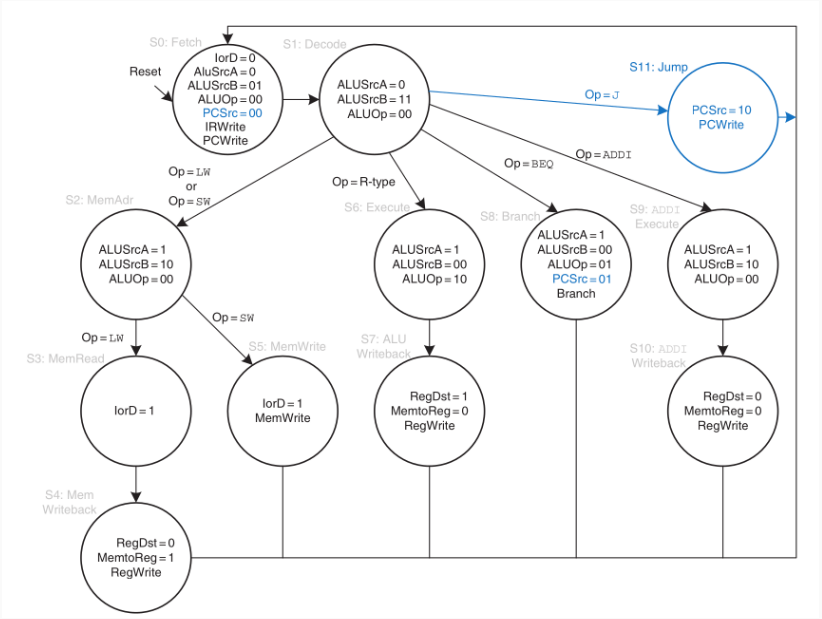
We are making a custom processor for use in real-time low-power DSP. It will be based on the multi-cycle MIPS design, but requires a single instruction to multiply in-memory signal values with in-register filter weights [1]. Explicitly, we want

```
memmul rt imm(rs):
M[R[rs+SignExt(imm)]] :=
M[R[rs+SignExt(imm)]] * R[rt]
```

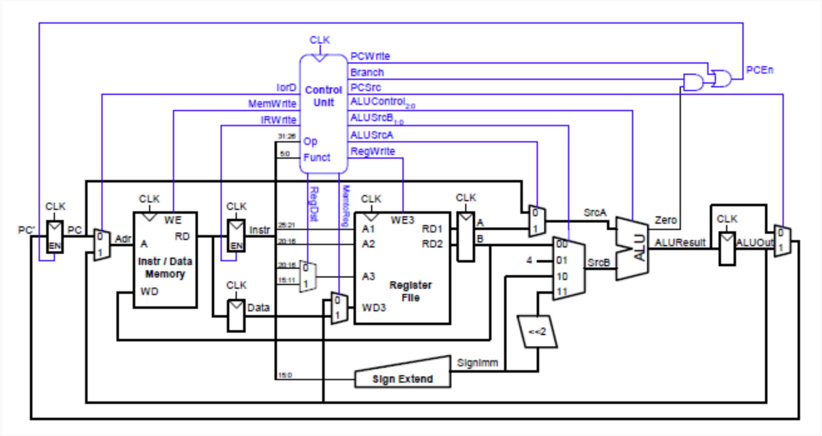
Do we need to change anything in the datapath? In the control?

Assume there is a 2-cycle multiplier availible in the ALU.

[1] extra for experts: how would you build convolution around this operation?



S0: Fetch
IorD = 0
AluSrcA = 0
ALUSrcB = 01
ALUOp = 00
PCSrc = 00
IRWrite
PCWrite
Reset
S1: Decode
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
S11: Jump
Op = J
PCSrc = 10
PCWrite
Op = ADDI
Op = BEQ
Op = R-type
Op = LW
or
Op = SW
S2: MemAdr
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
S6: Execute
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
S8: Branch
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCSrc = 01
Branch
S9: ADDI
Execute
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
Op = SW
Op = LW
S3: MemRead
IorD = 1
S5: MemWrite
IorD = 1
MemWrite
S7: ALU
Writeback
RegDst = 1
MemtoReg = 0
RegWrite
S10: ADDI
Writeback
RegDst = 0
MemtoReg = 0
RegWrite
S4: Mem
Writeback
RegDst = 0
MemtoReg = 1
RegWrite


CLK
PCWrite
Branch
PCSrc
ALUControl2:0
ALUSrcB1:0
ALUSrcA
RegWrite
PCEn
IorD
MemWrite
IRWrite
Control Unit
Op
Funct
31:26
5:0
RegDst
MemtoReg
PC'
PC
EN
Adr
A
WE
RD
Instr / Data Memory
WD
Instr
Data
25:21
20:16
15:11
15:0
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
A
B
SrcA
SrcB
4
00
01
10
11
<<2
SignImm
Sign Extend
ALU
Zero
ALUResult
ALUOut
0
1

Memory in your processor (Lab4)

```
// Instruction memory (already implemented)
module imem(input    [5:0]  a,
            output   [31:0] rd);

  reg [31:0] RAM[63:0];

  initial
    begin
      $readmemh("memfile.dat",RAM);
      // initialize memory with test program.
    end

  assign rd = RAM[a]; // word aligned
endmodule
```

Memory in your processor (Lab4)

When can you read from this? What can you write to it?

When do you need to read/write from the data memory? The regfile?

The address space isn't a full 32 bits. What addressing modes are affected?