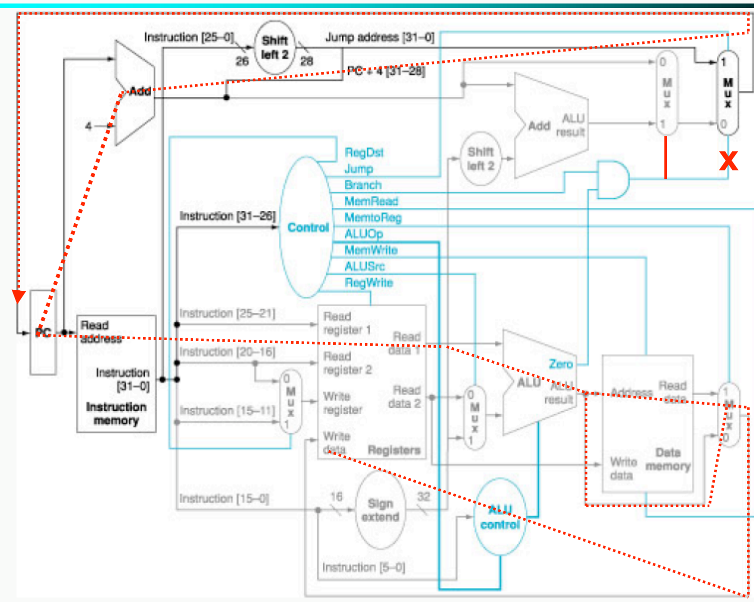
CSE 30321
Computer Architecture I

Lecture 17 - Multi Cycle Machines

Michael Niemier
Department of Computer Science and Engineering

Single cycle Control Implementation



How to Determine Cycle Length?

- Calculate cycle time assuming negligible delays except:
  - memory (2ns), ALU and adders (2ns), register file access (1ns)
  - R-type: max {mem + RF + ALU + RF, Add}
    = 6ns
  - LW: max{mem + RF + ALU + mem + RF, Add} = 8ns
  - SW: max{mem + RF + ALU + mem, Add} = 7ns
  - BEQ: max{mem + RF + ALU, max{Add, mem + Add}}
    = 5ns

Some Observations

- Datapath:
  - How many times is each component used during an instruction execution?
  - Components can be combined by overlapping different instruction types
    - Register file by all instruction types
    - How about ALU?
    - How about sign-extension unit?
- Control:
  - For each type of instruction, identify control signals for each datapath component involved
  - Control signals are generated from the instruction opcode (instr[31:26])

Single-Cycle Implementation

- Single-cycle, fixed-length clock:
  - CPI = 1
  - Clock cycle = propagation delay of the longest datapath operations among all instruction types
  - Easy to implement
- Single-cycle, variable-length clock:
  - CPI = 1
  - Clock cycle = Σ (%(type-i instructions) * propagation delay of the type-i instruction datapath operations)
  - better than the previous one but impractical to implement
- Disadvantages:
  - What if we have floating-point operations?
  - How about component usage?

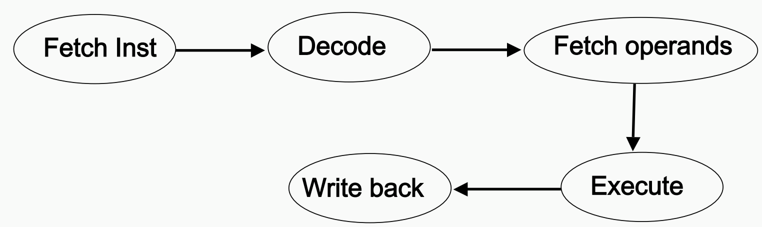
Multiple Cycle Alternative

- Break an instruction into smaller steps
- Execute each step in one cycle
- Execution sequence:
  - Balance the amount of work to be done, why?
  - Restrict each cycle to use only one major functional unit, why?
  - At the end of a cycle
    - store values for use in later cycles, why?
    - introduce additional “internal” registers
- The advantages:
  - Cycle time is much shorter
  - Different instructions take different number of cycles to complete
  - Allows a functional unit to be used more than once per instruction

Multiple-Cycle Implementation

- Datapath
  - Component sharing: ALU, Instruction/Data memory
    - ALU used to compute address and to increment PC
    - Memory used for instruction and data
  - Additional elements: MUX’s, Instr Register, Target Register
    - If a value needs to be alive during multiple cycles, it should stay unchanged during the whole time.
- Control:
  - Needed for each datapath element during each clock cycle

What to be Done for Each Instruction?



- How many cycles should the above take?
- You are the architect so you decide!
- Less cylces => more to be done in one cycle

Five Step Execution

1. Instruction Fetch (Ifetch):
   - Fetch instruction at address ($PC)
   - Store the instruction in register IR
   - Increment PC
2. Instruction Decode and Register Read (Decode):
   - Decode the instruction type and read register
   - Store the register contents in registers A and B
   - Compute new PC address and store it in ALUOut
3. Execution, Memory Address Computation, or Branch Completion (Execute):
   - Compute memory address (for LW and SW), or
   - Perform R-type operation (for R-type instruction), or
   - Update PC (for Branch and Jump)
   - Store memory address or register operation result in ALUOut

Five Step Execution (cont'd)

4. Memory Access or R-type instruction completion (MemRead/RegWrite/MemWrite):
   - Read memory at address ALUOut and store it in MDR
   - Write ALUOut content into register file, or
   - Write memory at address ALUOut with the value in B
5. Write-back step (WrBack):
   - Write the memory content read into register file

- Number of cycles for an instruction:
  - R-type: 4
  - lw: 5
  - sw: 4
  - Branch or Jump: 3

Some Simple Questions

- How many cycles will it take to execute this code?

```
      lw  $t2, 0($t3)
      lw  $t3, 4($t3)
      beq $t2, $t3, Label #assume branch is not taken
      add $t5, $t2, $t3
      sw $t5, 8($t3)
Label: ...
```

5+5+3+4+4=21

- What is being done during the 8th cycle of execution?

Compute memory address: 4+$t3

- In what cycle does the actual addition of $t2 and $t3 takes place? 16

Step 1: Instruction Fetch

- Use PC to fetch instruction and put it in the Instruction Register.
- Increment the PC by 4 and put the result back in the PC.
- How about express this in RTL?

IR=Mem[PC], PC=PC+4

- What is the advantage of updating the PC now?
- Basic principle: do it ASAP!

Step 2: Decode and Register Read

- Read registers rs and rt in case we need them
- Compute the branch address in case the instruction is a branch
- RTL:

```
A = RF[IR[25:21]],
B = RF[IR[20:16]],
ALUOut = PC +(sign-extend(IR[15-0]))<<2
```

- Did we set any control lines based on the instruction type?

Step 3 Execute (Instruction Dependent)

- ALU is performing one of three functions, based on instruction type
- RTL
  - Memory Reference:

    ```
    ALUOut = A + sign_ext(IR[15:0]);
    ```
  - R-type:

    ```
    ALUOut = A op B;
    ```
  - Branch:

    ```
    if (A=B) then (PC = ALUOut);
    ```


Step 4 RegWrite/MemRead

- Loads and stores access memory

```
MDR = Mem[ALUOut];
   or
Mem[ALUOut] = B;
```

- R-type instructions finish

```
RF[IR[15:11]] = ALUOut;
```


Step 5: Write-Back

- Which type of instruction needs this?
- RTL

```
RF[IR[20:16]]= MDR;
```

- What about all the other instructions?

RTL Description: Put All Together (1)

Ifetch: -> Decode,
    IR = Mem[PC], PC = PC + 4;

Decode: ->Execute,
    A= RF[IR[25:21]], B= RF[IR[20:16]],
    ALUOut = PC + Sign_Ext(IR[15:0]) << 2);

Execute:
    if (opcode=lw) or (opcode=sw) then -> MRead/RegWrite,
        ALUOut = A + Sign_Ext(IR[15:0]);
    if (opcode="R-type") then -> MRead/RegWrite,
        ALUOut = A op B;
    if (opcode=branch) then -> Ifetch,
        if (A=B) then PC= ALUout;
    if (opcode=jump) then -> Ifetch,
        PC=PC[31:28]||IR[25:0]||00;

RTL Description: Put All Together (2)

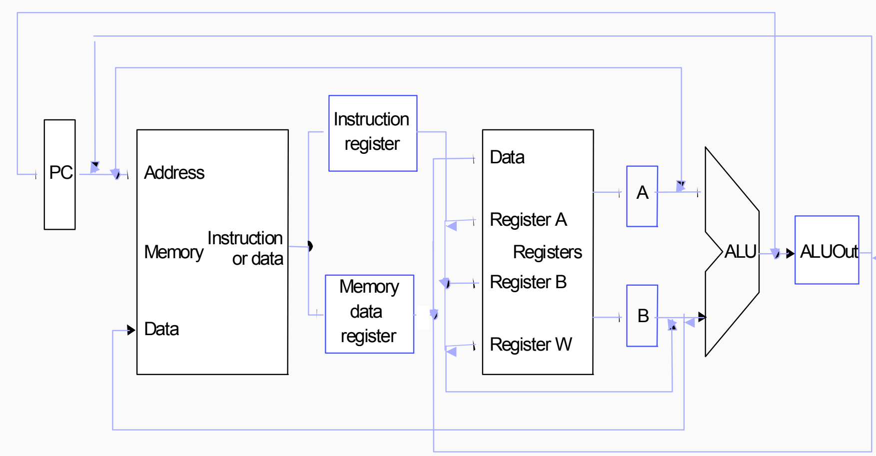
MRead/RegWrite:
    if (opcode=lw) then -> WriteBack,
        MDR = Mem[ALUOut];
    if (opcode=sw) then -> Ifetch,
        Mem[ALUOut] = MDR;
    RF[IR[15:11]] = ALUOut, ->Ifetch;

WriteBack:
    Mem[ALUOut] = MDR, ->Ifetch;

Execution Sequence Summary

Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	IR = Mem[PC], PC = PC + 4			
Instruction decode/register fetch	A =RF [IR[25:21]], B = RF [IR[20:16]], ALUOut = PC + (sign-extend (IR[1:-0]) << 2)			
Execution, address computation, branch/ jump completion	ALUOut = A op B	ALUOut = A + sign-extend (IR[15:0])	if (A =B) then PC = ALUOut	PC = PC [31:28] | (IR[25:0]<<2)
Memory access or R-type completion	RF [IR[15:11]] = ALUOut	Load: MDR = Mem[ALUOut] or Store: Mem[ALUOut]= B		
Memory read completion		Load: RF[IR[20:16]] = MDR		

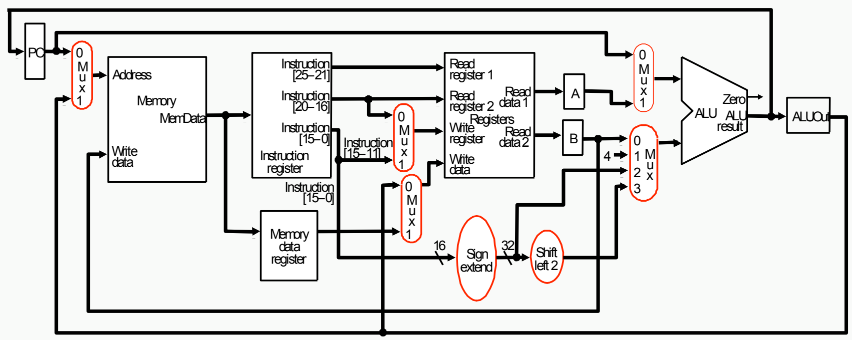
A Multiple Cycle Datapath



- Where do we need to insert mux's?
- Any other functional units?

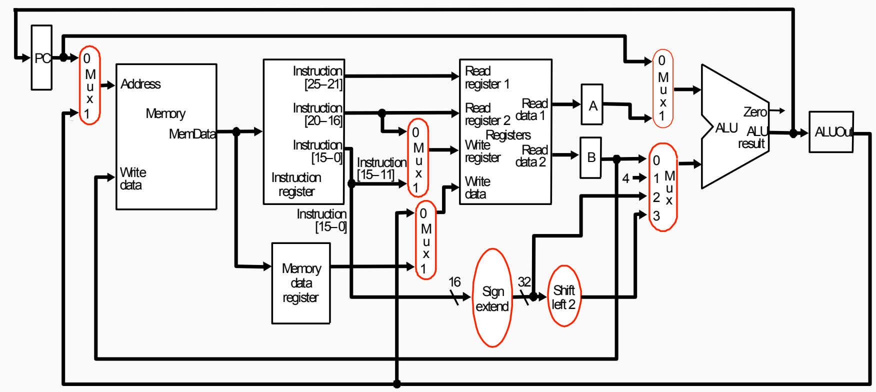
Multiple Cycle Design

- Break up the instructions into steps, each step takes a cycle
  - balance the amount of work to be done
  - restrict each cycle to use only one major functional unit
- At the end of a cycle
  - store values for use in later cycles (easiest thing to do)
  - introduce additional "internal" registers

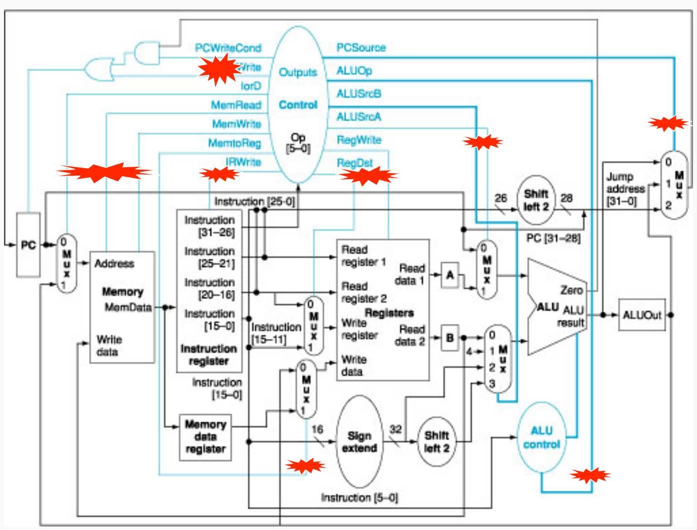


Exercise: Add a New Instruction

- Let's try "jal"
- RTL: PC = (PC+4)[3:0] || TargetAddr[25:0],
  RF[31] = PC + 4;



Control Signals



- **PC**: PCWrite, PCWriteCond, PCSource
- **Memory**: IorD, MemRead, MemWrite
- **Instruction Register**: IRWrite
- **Register File**: RegWrite, MemtoReg, RegDst
- **ALU**: ALUSrcA, ALUSrcB, ALUOp,

Implementing the Control

- Value of control signals is dependent upon:
  - what instruction is being executed
  - which step is being performed
- How to represent all the information?
  - finite state diagram
  - microprogramming
- Realization of a control unit is independent of the representation used
  - Control outputs: random logic, ROM, PLA
  - Next-state function: same as above or an explicit sequencer

Finite State Diagram