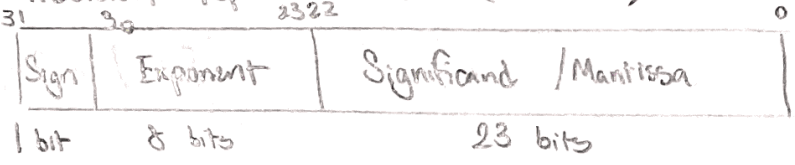
CS 61C Midterm Notes

Jeffrey Shen

Floating Point Representation

Single Precision Representation (32 bit)



$$Value = (-1)^{sign} \times 2^{exp + bias} \times (1 + Significand)$$

Exponent	Significand	Object
0	0	0
0	nonzero	Denorm
1-254	any	+/- floating pt #
255	0	+/- ∞
255	nonzero	NaN

Denorm: $Value = (-1)^{sign} \times 2^{bias+1} \times (0.\ significand)$

- Allows smallest value $a = 2^{-149}$

RISC-V

Instructions: [operation] [destination], [source1], [source2]

- RISC-V is little Endian - lowest byte on right w/ lowest address
- Program Counter: Internal Register holding byte address of next instruction
- Pseudo-instructions: shorthand syntax for common assembly idioms (mv, li, nop)

Function Calls

Steps

1. Put arguments in registers for function (a0-a7)
2. Transfer control to function (jal)
3. Acquire local storage resources (prologue)
4. Perform desired task of function
5. Put return value into register and release local storage
6. Return control to point of origin (ret)

Ex

Prologue:
```
addi  sp, sp, -8
sw    s1, 4(sp)
sw    s0, 0(sp)
```

Epilogue:
```
lw    s0, 0(sp)
lw    s1, 4(sp)
addi  sp, sp, 8
jr    ra
```

Calling Convention

In function (callee), at prologue, epilogue

Save:
- sp
- S registers (s0-s11)

Before calling function (caller), save:
- ra
- t registers (t0-t6)
- a registers (a0-a7)

Instruction Formats

R: register register arithmetic

I: immediate arithmetic, loads

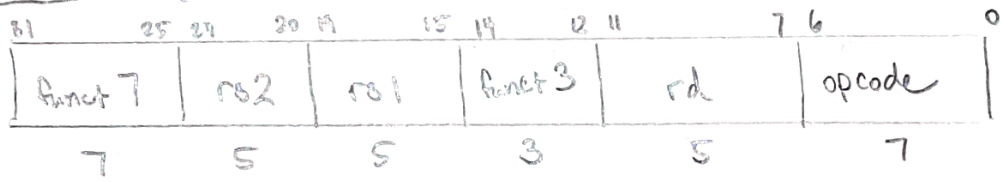
S: Stores

B: Branches

U: Upper Immediate

J: Jumps

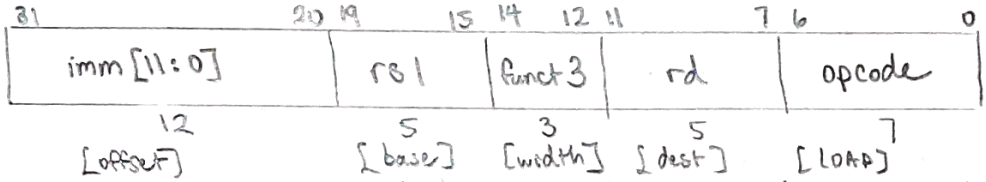
R-Format (Register)



Operations : add, sub, sll, slt, sltu, xor, srl, sra, or, and

I-Format (Immediate)

Immediate values (-2048, 2047)

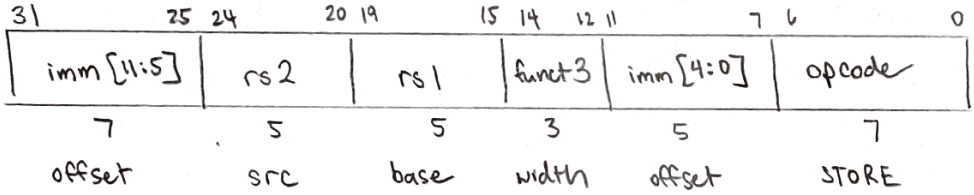


Operations : Immediates : addi, slti, sltu, xori, ori, andi, slli, srli, srai

Loads : lb, lh, lw, lbu, lhu

Jumps : jalr

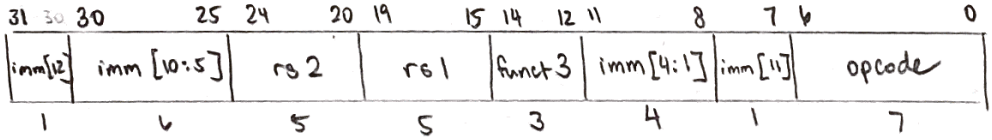
S-Format (Stores)



sw src, offset(base)

Operations : sb, sh, sw

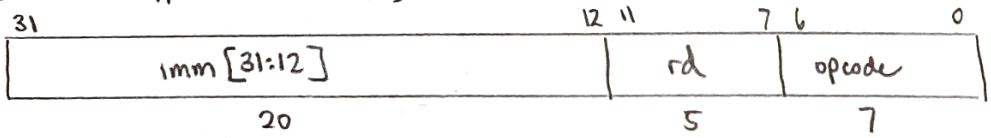
B-Format (Branches)



Implicit imm[0] = 0

Operations: beq, bne, blt, bge, bltu, bgeu

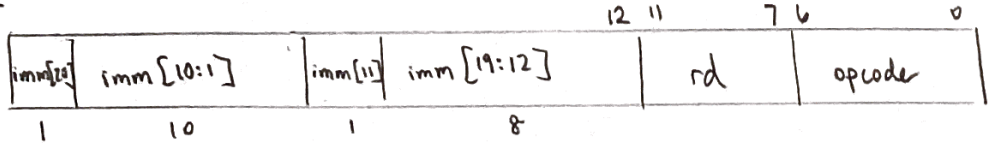
U-Format (Upper Immediate)



li will handle case of sign extend

Operations : lui, auipc

J-Format (Jumps)



jal: rd = PC+4
PC = PC + offset

jalr: rd = PC+4
PC = rs + imm

Operations : jal

implicit 0

Compiler, Assembler, Linker, Loader (CALL)

- Interpret a high level language when efficiency not critical to exec other programs
- Translate to lower level language to increase performance, hide source

Compiler

High level language → Assembly Language (C → RISC-V)

- Translates language

Assembler

Assembly language → object files (RISC-V → file.o)

- Replace pseudo-instructions
- Two passes to determine offset between jumps
- Reads and uses directives (.text, .data, .string, .globl)
- Produces relocation tables (to fill in later when you link in other files) and symbol tables (list of items in this file that may be used/referenced)

Linker

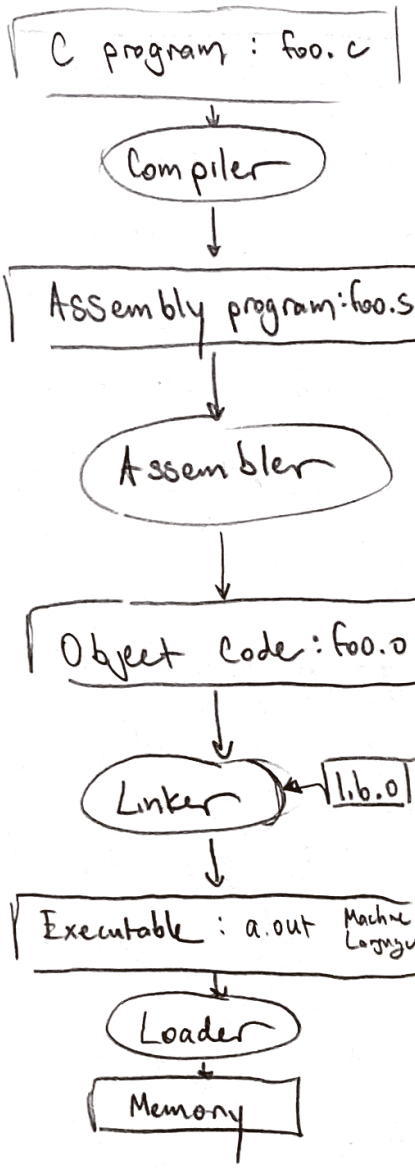
object files → executable code (file.o → a.out)

- Fulfills missing labels in relocation symbol tables
- Combines object files into binary executable

Loader

executable file → program run

- Reads header and creates memory space, set up for execution



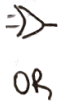
Combinational Logic

NOT

flips input

AND

1: both inputs 1
0: otherwise

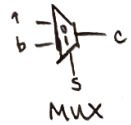


OR

1: at least one input 1
0: otherwise

XOR

1: inputs different
0: otherwise



MUX

pick among inputs
S input selects one of 2^S

Boolean Algebra

+ : OR

* : AND

$\overline{\ }$: NOT

$x + yz = (x+y)(x+z)$

$xy + x = x$

$\overline{x \cdot y} = \overline{x} + \overline{y}$

$(x+y)\,x = x$

$\overline{(x+y)} = \overline{x} \cdot \overline{y}$

distributive

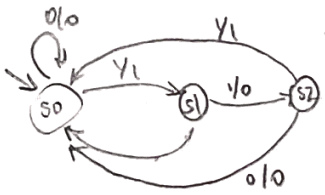
uniting theorem

DeMorgan's Law

Canonical Forms : sum of products

- $y = \bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c} + a\bar{b}\bar{c} + abc$

Can find simplified using truth table



Finite State Machines : number of states, transitions

- State denoted as start state, one arrow for input
- X/Y where input X, then output Y following arrow

Sychronous Digital Systems

Registers : controlled by clock, storage object

Value updated at "rising edge of the clock" otherwise constant

- Setup time : time before rising edge where input must be stable
- Hold time : time after rising edge where input must be stable
- Clock-to-Q Delay : time it takes for register's input to become its output after rising edge

Critical Path : longest delay between state elements

$$t_{clk} \geq t_{clk\text{-}to\text{-}q} + t_{logic} + t_{setup}$$

t_{logic} ↑ longest logical path

Hold time requirement : $t_{clk\text{-}to\text{-}q} + t_{logic\ shortest} > t_{hold}$

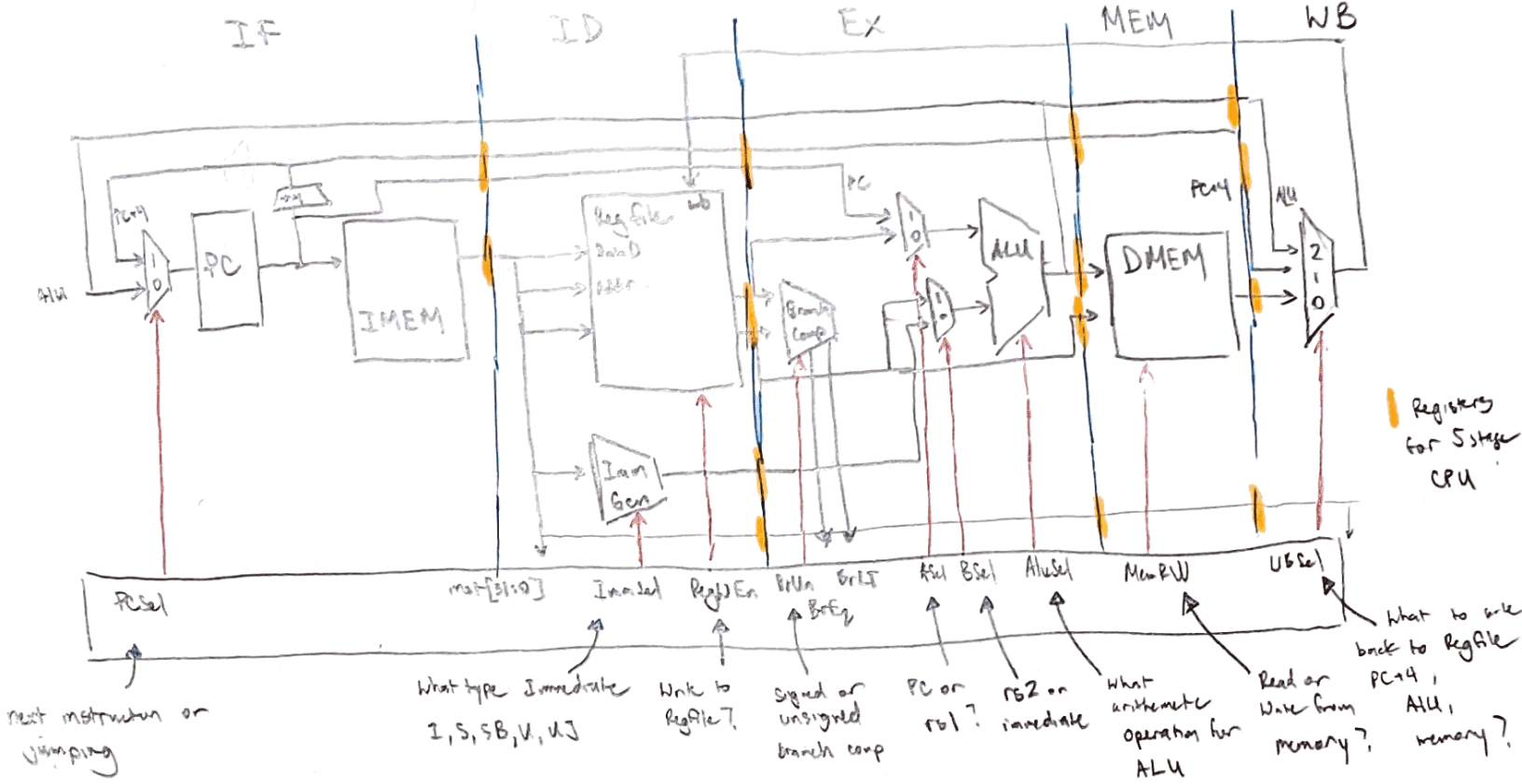
Datapath

Processor (CPU) : implemented directly in hardware (ISA)

- Datapath : contains hardware to perform operations
- Control : part of processor telling datapath what needs to be done

5 Stages of Data path

1. Instruction Fetch (IF)
2. Instruction Decode (ID)
3. Execute - ALU (EX)
4. Memory Access (MEM)
5. Write Back to Register (WB)



Critical Path for Single cycle = $t_{clk\text{-}to\text{-}q} + t_{IMEM} + t_{RegF} + t_{mux} + t_{ALU} + t_{DMEM} + t_{mux}$

Clock frequency = $\frac{1}{\text{critical path}}$

Pipelining: Add registers between stages to speed up

Latency: time for 1 instruction to finish

Throughput: # instructions processed per unit of time

-Pipeling increases throughput but also latency

Pipelining Hazards

1. Structural Hazards
   - more than one instruction needs to use resource
   caused by: register file ID, WB, Memory IMEM, DMEM
   solved by: hardware
2. Data Hazards
   - data dependencies between instructions
   caused by: instruction reads register before prev finished writing
   solved by: 1. Forwarding: result of EX, MEM sent to EX for next
   2. Stalls (lw) nop to stall

3. Control Hazards
- Jump and branch and unsure of next PC
caused by: jump and branch instruction
solved by: 1. Branch prediction predict where to go from prev
kill instruction if not taken
2. Stall

Double Pumping: allows writing and reading from reg file in one stage