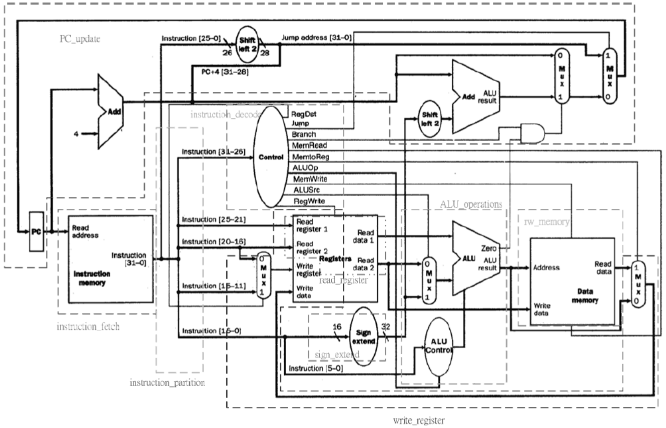
Final Project:
MySPIM



Slides by **Jerrett Longworth**
All Original Credit to **Sarah Angell**

CDA 3103, Spring 2024
Revision 2

Project Overview

- You are to simulate a MIPS processor in C.
- You are already given a skeleton of how the emulator should function.
- Just fill in the functions.

Misconceptions

- You do not need to make any changes to spimcore.c or spimcore.h.
- You do not need to worry about input or output.
  - This implies printf() and scanf() statements.
  - Adding these may break test cases.
- You do not need to convert between hexadecimal and decimal.

Development Environment

- You must compile with gcc.
  - Compile with: gcc -o spimcore spimcore.c project.c
  - Run with: ./spimcore <filename>.asc
- You can use any environment to compile and run, but you must test on Eustis.

MySPIM Controls

- This is already provided for you from spimcore.c. It is like a “shell” around your project.c.

r	Register	Display register contents.
m	Memory	Display memory contents.
s	Step	Attempt to run one instruction, located at the current PC.
c	Continue	Attempt to run all instructions, starting at the current PC.
h	Halt	Check to see if the simulation has halted.
g	Controls	Display the most recent control signals.
q	Quit	Close the simulation.

How to Start

- Suggestion: Implement functions in order of the datapath.
  - instruction_fetch(…)
  - instruction_partition(…)
  - instruction_decode(…)
  - And so on.
- You can check the output of one function before making the next.
- (GDB can be your friend!)

instruction_fetch

- `int instruction_fetch(unsigned PC,unsigned *Mem,unsigned *instruction)`

- Mem has already been populated, and PC will be initialize at the starting address (0x4000).
- Check for word alignment.
- Use “PC >> 2” to get the array index.

instruction_partition

- `void instruction_partition(unsigned instruction, unsigned *op, unsigned *r1,unsigned *r2, unsigned *r3, unsigned *funct, unsigned *offset, unsigned *jsec)`

- unsigned op // instruction [31-26]
- r1 // instruction [25-21]
- r2 // instruction [20-16]
- r3 // instruction [15-11]
- funct // instruction [5-0]
- offset // instruction [15-0]
- jsec // instruction [25-0]

instruction_decode

- `int instruction_decode(unsigned op,struct_controls *controls)`

-
```
typedef struct
{
   char RegDst;
   char Jump;
   char Branch;
   char MemRead;
   char MemtoReg;
   char ALUOp;
   char MemWrite;
   char ALUSrc;
   char RegWrite;
}struct_controls;
```

read_register

- `void read_register(unsigned r1,unsigned r2,unsigned *Reg,unsigned *data1,unsigned *data2)`

sign_extend

- `void sign_extend(unsigned offset,unsigned *extended_value)`

- The 16th bit is the sign bit.
- Recall during the partitioning function the offset has all zeros in the upper 16 bits.

ALU_operations

- `int ALU_operations(unsigned data1,unsigned data2,unsigned extended_value,unsigned funct,char ALUOp,char ALUSrc,unsigned *ALUresult,char *Zero)`

- This sets the parameters for the ALU's A, B, and ALUControl inputs.
- If this is an R-type instruction, look at funct.
- Call ALU() function at the end.

rw_memory

- `int rw_memory(unsigned ALUresult,unsigned data2,char MemWrite,char MemRead,unsigned *memdata,unsigned *Mem)`

- If MemWrite = 1, write to memory.
- If MemRead = 1, read from memory.

write_register

- `void write_register(unsigned r2,unsigned r3,unsigned memdata,unsigned ALUresult,char RegWrite,char RegDst,char MemtoReg,unsigned *Reg)`

- If RegWrite == 1 and MemtoReg == 1, then bring data from memory.
- If RegWrite == 1 and MemtoReg == 0, then bring data from ALUresult.

PC_update

- `void PC_update(unsigned jsec,unsigned extended_value,char Branch,char Jump,char Zero,unsigned *PC)`

- PC = PC + 4.
- Take note of Branch and Jump.
- Zero and Branch tell to branch or not.
- For jumps: Left shift bits of jsec by 2 and use upper 4 bits of PC.

Additional Hints (1)

- Accessing memory
  - Mem[0] is an unsigned int, in other words, the whole word.
  - This means that given an address, like the PC, you will need to shift the address to the right by 2 to form the index for the Mem structure.
  - Example: The 32-bit word at addresses 0x4000-0x4003 would be Mem[1000]

Additional Hints (2)

- Example of isolating bits in an unsigned int via bitwise AND and shift in C:
  - `int op = (inst & 0xFC000000) >> 26;`

- The input file is a series of hexadecimal numbers.
  - Don’t write anything to parse the file. It is already done for you.
  - The numbers in the Mem structure are already at the proper addresses.

Additional Hints (3)

- The input file is a series of hexadecimal numbers.
  - Understand what “unsigned int instruction = MEM(PC)” would give you. (See line 14 of spimcore.c)
  - MEM(PC) *IS* the 32-bit instruction. You don’t need to convert it to hexadecimal or binary, or do any adjustments to the instruction.