In this lecture, I want a lot of help and participation. You now have the fundamental knowledge to design a processor. Let’s build a simple one on paper. You’ll be programming a slightly more complex processor in next week’s lab assignment.

- Micro-controller based design
- In this lecture, I want a lot of help and participation
- You now have the fundamental knowledge to design a processor
- Let’s build a simple one on paper
- You’ll be programming a slightly more complex processor in next week’s lab assignment.

- Already understand building FSMs
- Can use array of latches to store multiple bits: register
- Consider simple processor, called RSE (Rob’s simplified example)

- All registers are 8-bit
- Four general-purpose registers, A, B, C, and D
  - Used to do computation
  - Program counter PC
- Stack pointer SP (sometimes called TOS for top of stack), which may also be used as a general-purpose register
- ALU capable of adding (0) and subtracting (1)

- add $R_D$, $R_{S1}$, $R_{S2}$
- sub $R_D$, $R_{S1}$, $R_{S2}$

Do computation on source registers and put result in destination register

- ldm $R_D$, $[R_S]$
  - Load from memory location indicated by the source register into destination register
- stm $[R_D]$, $R_S$
  - Store to memory location indicated by the destination register from source register
- ldi $R_D$, $I$
  - Load immediate into destination register
- ldpc $R_S$
  - Load from program counter to destination register

- blz $R_T$, $R_C$
  - Set PC to $R_T$ if $R_C < 0$
- bzc $R_T$, $R_C$
  - Set PC to $R_T$ if $R_C = 0$
Instruction encoding

- How many instructions?
- Worst-case operands?
  - 3 registers (each how many bits?)
  - 1 register and 1 immediate
  - To pack or not to pack?

Initialization

- Chip has reset line
- Set PC to byte 2
- Start running...

Memory

- Acts like a collection of byte-wide registers
- Address using a decoder
- Can put other devices at some memory locations
  - Memory-mapped input-output
- Can also use special-purpose output instructions or registers
- Let's build some from D flip-flops
- Multiplexing address and data lines?

Program counter

Every clock tick the processor
- Fetches an instruction from the memory location pointed to by PC
- Decodes the instruction
- Fetches the operands
- Executes the instruction
- Stores the results
- Increments the program counter
- Can jump to another code location by moving a value into the PC

Example high-level code

Sum up the contents of memory locations 2–6
- $A = 0$
- For $B$ from 2 to 6
- $A = A + [B]$

Example low-level code

Sum up the contents of memory locations 2–6

2. $A = 0$
4. $B = 2$
6. $C = [B]$ (loop start point)
8. $A = A + C$
10. $B = B + 1$
12. $C = 6$ (loop start)
16. If $B \leq 6$ ($B < 7$) branch to $C$
   - $D = 7$ — sub $D$, $B$, $D$ — blz $C$, $D$
   - (Done)

Error conditions

- What happens on overflow or underflow?
- Special register?
- Special value associated with each register?
- Single-instruction compare and branch?
- Advantages and disadvantages of each?

Assemble to our encodings

- After assembling, can put program contents into memory, starting at byte 2
- Compiling from higher-level languages also possible
Example high-level code

Sum up the contents of memory locations 2-6

1. \( i = 0 \)
2. For \( j \) from 2 to 6
   \[ i = i + [j] \]

Lesson

- With only a few registers and instructions, powerful actions are possible
- Less time and power efficient than special-purpose hardware design
- Instruction processors are flexible
- Allows massive use of a single type of IC
- Assembly is painful
- However, much better than doing hardware design
- Compilation also possible

Today’s topics

- Architecture
- Assembly
- Compilation
- PIC16C74A

Assigned reading

- Refer to Chapter 7 and 8
- Read Sections 9.1–9.7, 10.1–10.6, 10.8

Computer geek culture references

- Building multicontroller-based devices for the fun of it
- http://www.bdmicro.com
- http://members.home.nl/bzijlstra/
- Etc.