VE370 Midterm RC

(part I)

2020.10.17

Zhengfei He

General Tips

- Difficulty: homework > exam > slides
- Suggested Review order: slides > homework

Problem **Guessing** (not sure)

- **CPU performance**
  - Time calculation and comparison
- **Assembly**
  - Instructions' characteristics, registers' characteristics
  - Write MIPS assembly program following function calling convention
- **Single cycle CPU**
  - Data flow in the CPU based on an instruction
- **Pipeline CPU (focus)**
  - Control signals flow in a pipepline CPU based on a set of instructions
- **Data hazards**
  - Where, when, how
  - solutions

CPU performance

- CPU Time = CPU Clock Cycle per program * Clock Cycle Time
  = CPU Clock Cycles/ Clock Rate

- Clock Cycles = Instruction Count (IC) * Clock Cycle per Instruction(CPI)

- CPU Time = Instruction Count (IC) * CPI * Clock Cycle Time
  = IC *CPI / Clock Rate

- CPU Time = (Instruction /Program) * (Clock cycles / Instruction) * (Seconds/Clock cycle)

Assembly

- Use MIPS reference card
- Do not create MIPS instructions

Some important information

1. hexadecimal number must have 0x at the beginning, or it will be viewed as a decimal number
2. MIPS register $0 has constant value 0 cannot be overwritten
3. MIPS memory is byte-addressable
4. word address must be a multiple of 4
5. Signed number.Bit 31 is sign bit, 1 for negative numbers,0 for non-negative numbers
6. difference between lb and lbu, lh and lhu
7. Big Endian: Most significant byte at least address of a word
   Little Endian:Least significant byte at least address of a word
8. how to load 32 bit number to a register (lui and ori)

Example

For the number 0x2080C050

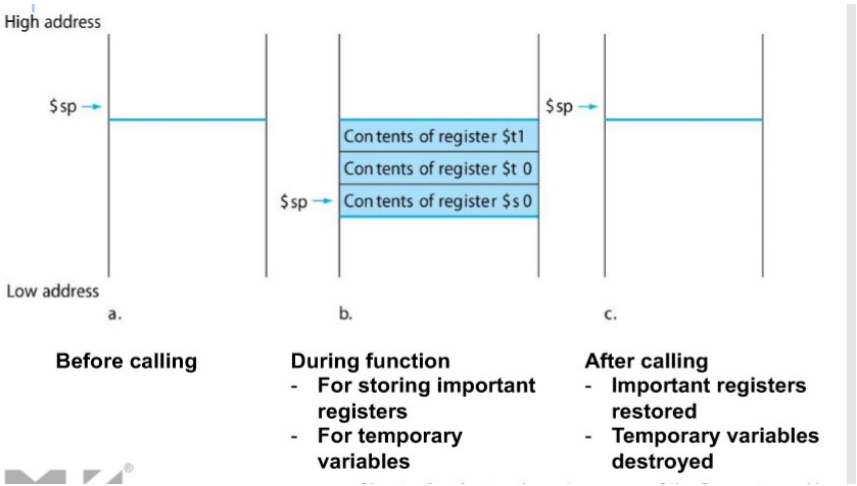
Big Endian

ADDRESS	0XFFFF0000	0XFFFF0001	0XFFFF0002	0XFFFF0003
Content	20	80	C0	50

Little Endian

ADDRESS	0XFFFF0003	0XFFFF0002	0XFFFF0001	0XFFFF0000
Content	20	80	C0	50

Function calling convention



- Before the function is called
  - Pass arguments to $a0-$a3
    - more arguments on stack
  - Save registers that should be saved by caller,
    - such as $a0-$a3 (non-leaf function), $t0-$t9 (if necessary)
  - `jal`
- Before function starts executing
  - Allocate memory of frame's size
    - by moving $sp downwords for frame's size
  - Save registers that should be saved by the function in the frame, before they are overwritten
    - $s0-$s7 (if to be used), $fp (if used), $ra (non-leaf function),
  - Establish $fp (if desired), $fp = $sp + frame's size – 4
- Before function finishes
  - If necessary, place function result to $v0, $v1
  - Restore registers saved by the function
    - Pop from frame
  - Destroy stack frame by moving $sp upword
  - `jr $ra`