ECE 571 – Advanced Microprocessor-Based Design Lecture 3

Vince Weaver

https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

9 September 2024

Announcements

- Homework #1 was posted, due Friday
- Let me know if you have any trouble logging in
- Yes, sometimes computer architecture research is a lot of boring measurements.
- Feel free to write scripts when gathering data, if it helps
- If need additional packages (within reason) let me know
- Note the disk on the server is not backed up

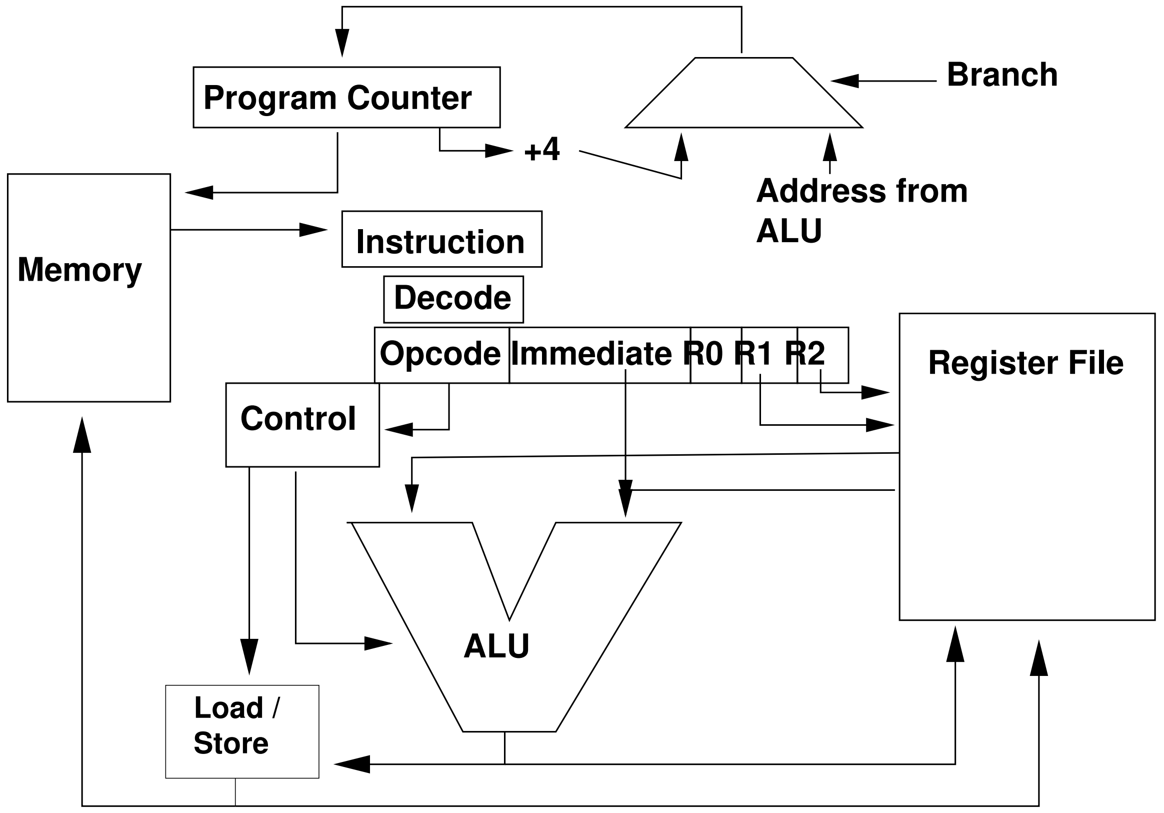
Some Background on the Homework

- bzip2 from SPEC2006 – compression program, by Julian Seward (also wrote Valgrind)
- modern replacement is xz (found in SPEC2017)

perf examples

- Went through the perf examples from the end of previous lecture notes again
- Not in as much detail as I'd like as wifi not working on the laptop

Simple CPU (again)



Simple CPU – Breakdown

- Program Counter / Instruction Pointer points to next instruction
  Increments each clock and loads next instruction from memory. If a branch instead loads new address if branch taken.
- Instruction is decoded. Opcode (says what type of instruction), Registers to use, possibly an immediate value.
- Opcode goes to control (usually a PLA) or microcode

that splits up signals to all the functional units and tells them what to do (what kind of operation, whether to read or write, to branch or not, etc)

- Source register values are read from register file and fed to ALU
- ALU does math/logic based on control
- Result written back to destination register on register file.
- If load or store instruction, then address calculated (often by ALU) and sent to memory. If load, value written to reg file, if store value to write sent out

- Once instruction is done, advance to next instruction.

CPU – Design decisions

- Used to be whole classes on this
- These days designing a new architecture from scratch is often discouraged
- It still occasionally happens

Instruction Set Architecture (ISA)

- List of instructions supported by an architecture
- Can guide design of processor
- Often stuck with old design decisions for backwards compatibility
- What instructions do you need?

ISA – baseline Integer Instructions

- ALU/Math: add/sub
  What about multiply/divide/modulus? Can you do in software? Division units are large/power-hungry
- ALU/Logic: and/or/xor
  Could you make any logic out of nand instructions? xor is turning complete on x86, can change any program into a series of xor
- Shifts/Rotates
- nop? often a pseudo-op

- memory: load/store?
  Can you operate direct on memory?
- compare (can be implemented with subtract and throw away result)
- branch: branch if zero/not zero
- unconditional branch
- function call (save return address)
- syscall?

RISC / CISC Discussion

- Simple decode. Load/store. Fixed instruction width. 3-operand.
- MIPS is classic RISC
- x86 is classic CISC (with complex instructions)
  Though internally x86 executes uops, RISC
- ARM (predication, auto-increment, barrel shifter)
  Called RISC but has complex instructions

RISC / CISC Example

Memory copy: Load a byte from pointer, store byte to another pointer, increment pointers, loop until counter counted down.

CISC	RISC
`rep movsb`	`ldb r0,[r1]`
	`add r1,r1,#1`
	`stb r0,[r2]`
	`add r2,r2,#1`
	`sub r3,r3,#1`
	`cmp r3,#0`
	`bne loop`

Note: if ARM32 can optimize a bit

Code Density

- Traditional RISC fixed 4-byte (32 bit) instructions
- CISC: x86 1-15 byte instructions (variable)
- Can have hybrid, Thumb/Thumb2 (2 and 4 byte)

Common ISAs

- x86
- ARM
- RISC-V
- Other RISC: MIPS, Power, Alpha, SPARC, PA-RISC, SH
- Other CISC: m68k, VAX, IBM 360
- itanium
- Lots more, Linux alone supports large number