B35APO: Architektury počítačů

Lekce 04. Central Processing Unit (CPU)

Pavel Píša
pisa@fel.cvut.cz

2. listopadu, 2022

Obsah

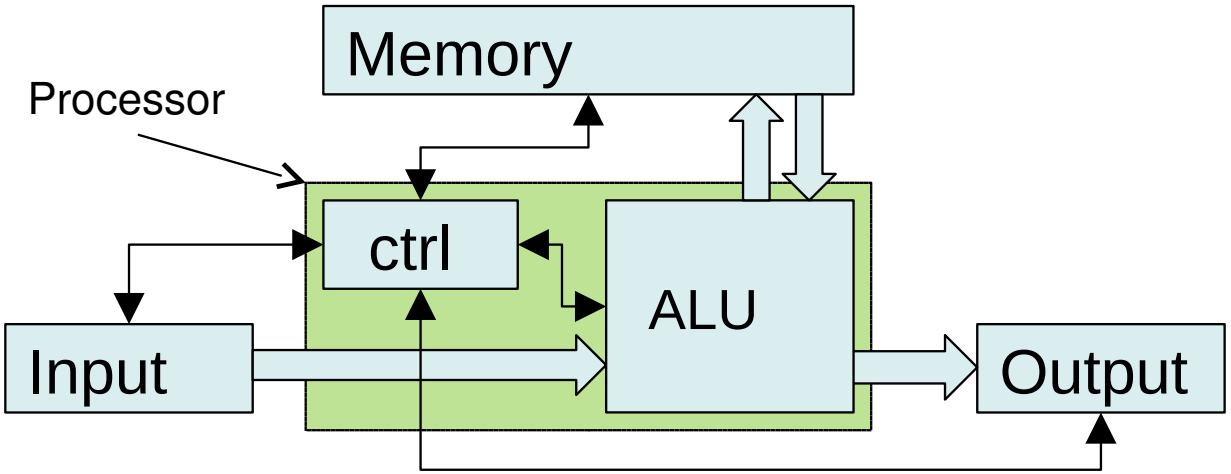
QtMips and QtRVSim – Origin and Development

- MipsIt used in past for Computer Architecture course at the Czech Technical University in Prague, Faculty of Electrical Engineering
- Diploma theses of Karel Kočí mentored by Pavel Píša Graphical CPU Simulator with Cache Visualization https://dspace.cvut.cz/bitstream/handle/10467/76764/F3-DP-2018-Koci-Karel-diploma.pdf
- Switch to QtMips in the 2019 summer semester
- Fixes, extension and partial internals redesign by Pavel Píša
- Switch to RISC-V architecture in 2022. Main work by Jakub Dupák 2021 Graphical RISC-V Architecture Simulator - Memory Model and Project Management
- https://dspace.cvut.cz/bitstream/handle/10467/94446/F3-BP-2021-Dupak-Jakub-thesis.pdf
- Alternatives:
  - RARS: IDE with detailed help and hints https://github.com/TheThirdOne/rars
  - EduMIPS64: 1x fixed and 3x FP pipelines https://www.edumips.org/

QtRVSim - Download

- Windows, Linux, Mac https://github.com/cvut/qtrvsim/releases
- Ubuntu https://launchpad.net/ qtrvsimteam/+archive/ubuntu/ppa
- Suse, Fedora and Debian
  https://software.opensuse.org/download.html?project=home
- Suse Factory TBD
- Online version https://dev.jakubdupak.com/qtrvsim/
- LinuxDays 2019 QtMips talk – Record of Interactive Session
  https://youtu.be/fhcdYtpFsyw
  https://pretalx.linuxdays.cz/2019/talk/EAYAGG/

John von Neumann



- 5 functional units – control unit, arithmetic logic unit, memory, input (devices), output (devices)
- An computer architecture should be independent of solved problems. It has to provide mechanism to load program into memory. The program controls what the computer does with data, which problem it solves.
- Programs and results/data are stored in the same memory. That memory consists of a cells of same size and these cells are sequentially numbered (address).

Computer based on von Neumann's concept

- Procesor/microprocessor
  - Control unit
  - ALU
- Memory - von Neumann architecture uses common memory, whereas Harvard architecture uses separate program and data memories
- Input/output subsystem
  - Input
  - Output

The control unit is responsible for control of the operation processing and sequencing. It consists of:

- registers – they hold intermediate and programmer visible state
- control logic circuits which represents core of the control unit (CU)

The most important registers of the control unit

- PC (Program Counter) holds address of a recent or next instruction to be processed
- IR (Instruction Register) holds the machine instruction read from memory
- Another usually present registers
  - General purpose registers (GPRs) may be divided to address and data or (partially) specialized registers
  - SP (Stack Pointer) – points to the top of the stack; (The stack is usually used to store local variables and subroutine return addresses)
  - PSW (Program Status Word)
  - IM (Interrupt Mask)
  - Optional Floating point (FPRs) and vector/multimedia regs.

The main instruction cycle of the CPU

FLAGS registr

1. Initial setup/reset – set initial PC value, PSW, etc.
2. Read the instruction from the memory
   - PC $\rightarrow$ to the address bus
   - Read the memory contents (machine instruction) and transfer it to the IR
   - PC+I $\rightarrow$ PC, where I is length of the instruction
3. Decode operation code (opcode)
4. Execute the operation
   - compute effective address, select registers, read operands, pass them through ALU and store result
5. Check for exceptions/interrupts (and service them)
6. Repeat from the step 2

Compilation: C Assembler Machine Code

```
/* ffs  as log2(x)*/
int x = 157;
int y = -1;

while(x != 0) {
  x = x / 2;
  y = y + 1;
}
```

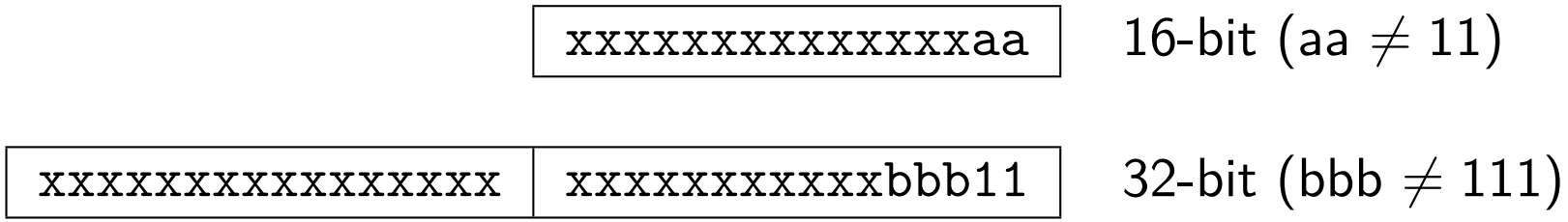
```
_start:
  addi a0, zero, 157
// int x = 157;
  addi t1, zero, -1
// int y = 0;
  beq a0, zero, done
// while(x != 0) {
loop:
  srli  a0, a0, 1      //
x = x / 2;
  addi t1, t1, 1       //
y = y + 1;
  bne a0, zero, loop   //}
done:
  addi a0, t1, 0
```

```
0x00000200
09d00513
0x00000204 fff00313
0x00000208
00050863
0x0000020c
00155513
0x00000210
00130313
0x00000214 fe051ce3
0x00000218
00030513
0x0000021c
00100073
```

Instruction Encoding Constrains

- Encode 256 combinations in 8-bits
- Opcode and address to directly operate on megabytes of ram does not fit
- Use some limited number of fast accessible registers or use stack concept
- 8 registers, 3 bits to encode, for two operand operations (a += b, mem[a] = b), 6 bits to select registers→only 4 operations in total
- 16 bit (65536), 16 registers, 256 two operands or 16 three operands (4 + 3 * 4 bits)
- 32 bit, 32 registers, three operands (17 + 3 * 5)
- But immediate for arithmetic and address usually required (CISC followup words, RISC uses limited ranges and some other mechanism)

RISC-V – Instruction Length Encoding



RISC-V Processor Registers

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	
x4	tp	Thread pointer	
x5-7	t0–2	Temporaries	
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–2	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	
pc	pc	Program Counter	
f0-31		Floating point	

Hardware realization of basic (main) CPU cycle

Single cycle CPU – implementation of the load instruction