Lecture 2:

Simple Computer Architecture I

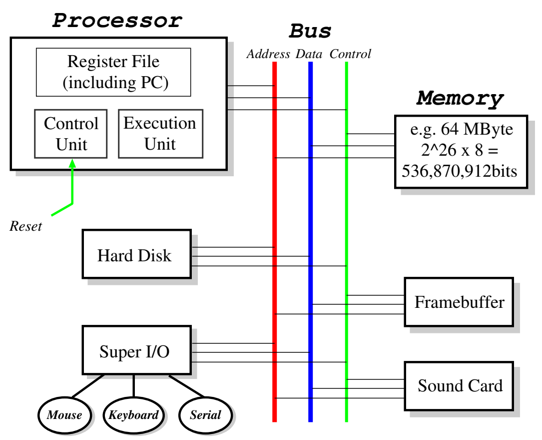
`www.cl.cam.ac.uk/Teaching/2001/OSFounds/`

Today's Lecture

Today we'll cover:

- What's inside a computer?
  - Registers,
  - Memory Hierarchy,
  - Control, Execution units
  - Fetch-Execute Cycle,
  - ALU.
- How do we store and manipulate numbers?
  - Sign and Magnitude,
  - Two's complement,
  - Arithmetic.

A (Simple) Modern Computer



- Processor (CPU): executes programs.
- Memory: stores both programs & data.
- Devices: for input and output.
- Bus: transfers information.

Registers and the Register File

R0	0x5A	R8	0xEA02D1F
R1	0x102034	R9	0x1001D
R2	0x2030ADCB	R10	0xFFFFFFFF
R3	0x0	R11	0x102FC8
R4	0x0	R12	0xFF0000
R5	0x2405	R13	0x37B1CD
R6	0x102038	R14	0x1
R7	0x20	R15	0x20000000

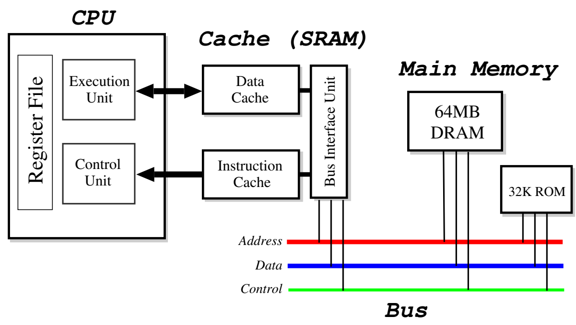
Computers are all about operating on information:

- information arrives into memory from input devices
- memory is a essentially large byte array which can hold any information we wish to operate on.
- computer *logically* takes values from memory, performs operations, and then stores result back.

In practice, CPU operates on **registers**:

- a register is an extremely fast piece of on-chip memory, usually either 32- or 64-bits in size.
- modern CPUs have between 8 and 128 registers.
- data values are *loaded* from memory into registers before being operated upon,
- and results are *stored* back again.

Memory Hierarchy

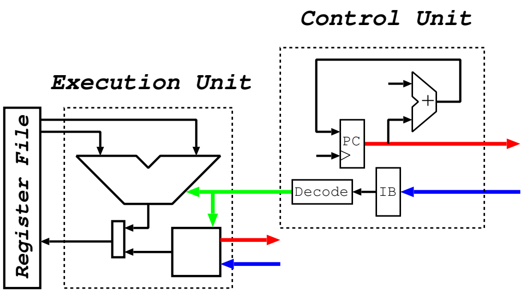


- Use **cache** between main memory and register: try to hide delay in accessing (relatively) slow DRAM.
- Cache made from faster SRAM:
  - more expensive, so much smaller
  - holds copy of subset of main memory.
- Split of instruction and data at cache level ⇒ "Harvard" architecture.
- Cache ↔ CPU interface uses a custom bus.
- Today have ~ 512KB cache, ~ 128MB RAM.

"Ideally one would desire an indefinitely large memory capacity such that any particular. . . word would be immediately available. . . We are. . . forced to recognize the possibility of constructing a **hierarchy** of memories, each of which has greater capacity than the preceding but which is less quickly accessible."

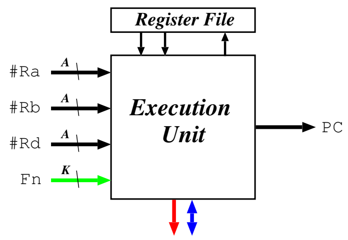
Burks, Goldstine & Von Neumann, 1946.

The Fetch-Execute Cycle



- A special register called PC holds a memory address; on reset, initialised to 0.
- Then:
  1. Instruction *fetched* from memory address held in PC into instruction buffer (IB).
  2. Control Unit determines what to do: *decodes* instruction.
  3. Execution Unit *executes* instruction.
  4. PC updated, and back to Step 1.
- Continues pretty much forever. . .

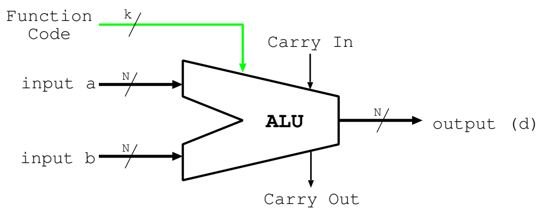
Execution Unit



- The "calculator" part of the processor.
- Broken into parts (**functional units**), e.g.
  - Arithmetic Logic Unit (ALU).
  - Shifter/Rotator.
  - Multiplier.
  - Divider.
  - Memory Access Unit (MAU).
  - Branch Unit.
- Choice of functional unit determined by signals from control unit.

Arithmetic Logic Unit

An N-bit ALU



- Part of the execution unit.
- Inputs from register file; output to register file.
- Performs simple two-operand functions:
  - a + b
  - a - b
  - a AND b
  - a OR b
  - etc.
- Typically perform *all* possible functions; use function code to select (mux) output.

Number Representation

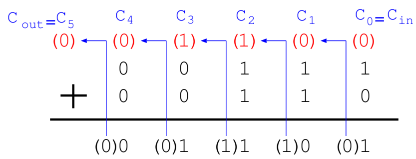
0000_2	0_{16}	0110_2	6_{16}	1100_2	C_{16}
0001_2	1_{16}	0111_2	7_{16}	1101_2	D_{16}
0010_2	2_{16}	1000_2	8_{16}	1110_2	E_{16}
0011_2	3_{16}	1001_2	9_{16}	1111_2	F_{16}
0100_2	4_{16}	1010_2	A_{16}	10000_2	10_{16}
0101_2	5_{16}	1011_2	B_{16}	10001_2	11_{16}

- a n-bit register $b_{n-1}b_{n-2}\dots b_1b_0$ can represent 2^n different values.
- Call b_{n-1} the most significant bit (msb), b_0 the least significant bit (lsb).
- Unsigned numbers: treat the obvious way, i.e. val $= b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \cdots + b_1 2^1 + b_0 2^0$, e.g. $1101_2 = 2^3 + 2^2 + 2^0 = 8 + 4 + 1 = 13$.
- Represents values from 0 to $2^n - 1$ inclusive.
- For large numbers, binary is unwieldy: use hexadecimal (base 16).
- To convert, group bits into groups of 4, e.g. $1111101010_2 = 0011|1110|1010_2 = 3EA_{16}$.
- Often use "$0x$" prefix to denote hex, e.g. $0x107$.
- Can use dot to separate large numbers into 16-bit chunks, e.g. $0x3FF.FFFF$.

Number Representation cont.

- What about *signed* numbers? Two main options:
- Sign & magnitude:
  - top (leftmost) bit flags if negative; remaining bits make value.
  - e.g. byte $10011011_2 \rightarrow -0011011_2 = -27$.
  - represents range $-(2^{n-1} - 1)$ to $+(2^{n-1} - 1)$, and the bonus value -0 (!).
- 2's complement:
  - to get $-x$ from x, invert every bit and add 1.
  - e.g. $+27 = 00011011_2 \Rightarrow$ $-27 = (11100100_2 + 1) = 11100101_2$.
  - treat $1000\dots000_2$ as -2^{n-1}.
  - represents range -2^{n-1} to $+(2^{n-1} - 1)$
- Note:
  - in *both* cases, top-bit means "negative".
  - both representations depend on n;
- In practice, all modern computers use 2's complement. . .

Unsigned Arithmetic



(we use 5-bit registers for simplicity)

- Unsigned addition: C_n means "carry":

```
    00101     5         11110    30
  + 00111     7       +  00111     7
  -------------        --------------
  0 01100    12        1 00101     5
  -------------        --------------
```

- Unsigned subtraction: $\overline{C_n}$ means "borrow":

```
    01100    12         00111     7
  + 11001    -7       +  10110   -10
  -------------        --------------
  1 00101     5        0 11101    29
  -------------        --------------
```

Signed Arithmetic

- In signed arithmetic, carry no good on its own. Use the **overflow** flag, $V = (C_n \oplus C_{n-1})$.
- Also have **negative** flag, $N = b_{n-1}$ (i.e. the msb).
- Signed addition:

```
     00101     5           01010    10
   + 00111     7       +   00111     7
  -------------        --------------
  0  01100    12       0   10001   -15
  -------------        --------------
     0                     1
```

- Signed subtraction:

```
     01010    10           10110   -10
   + 11001    -7       +   10110   -10
  -------------        --------------
  1  00011     3       1   01100    12
  -------------        --------------
     1                     0
```

- Note that in overflow cases the sign of the result is always wrong (i.e. the N bit is inverted).

Summary

You should now understand:

- Some details of simple computer architecture,
- The fetch-execute cycle,
- Binary and hexadecimal numbers,
- Representing signed numbers in binary, and
- Arithmetic with signed binary numbers.

Next lecture: Simple Computer Architecture II

Background Reading:

- Hennessy/Patterson:
  - Chapter 4 - Arithmetic
  - Chapter 7 - Memory