Lecture 2

Hexadecimal Numbers

Hexadecimal numbers are from 0 to F, where 10 is A, 11: B, 12: C, 13: D, 14: E, 15: F.

To convert a hexadecimal number to decimal we need to multiply each digit (from 0 to 15) by the appropriate power of 16, where the first digit should be multiplied by 16^0, the next one by 16^1, and so on.

Converting a hexadecimal number to binary and back is very easy, just take each hex digit and write it as a binary value from 0 to 15. Ex:

$$BAD_h = 101110101101_b$$

Where $B_h = 1011_b$, $A_h = 1010_b$, and $D_h = 1101_b$.

Binary	Decimal	Hexadecimal
0000	0	0
0001	1	1
0010	2	2
...	...	...
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

$$\mathrm{BAD}_{sixteen} = 1011\ 1010\ 1101_{two}$$

$$\mathrm{BAD}_{sixteen} = 11 \cdot 16^2 + 10 \cdot 16^1 + 13 \cdot 16^0$$

$$\mathrm{BAD}_{sixteen} = 2989$$

...

$$\mathrm{CAFE}_{sixteen} = 1100\ 1010\ 1111\ 1110_{two}$$

$$123_{sixteen} = 0001\ 0010\ 0011_{two} = 291$$

...

RGB colors:

$$\mathrm{FFAA88}_{sixteen} = \{255, 170, 136\}$$

Negative binary numbers

There are two main ways to convert negative numbers to binary: two’s complement, and sign and magnitude.

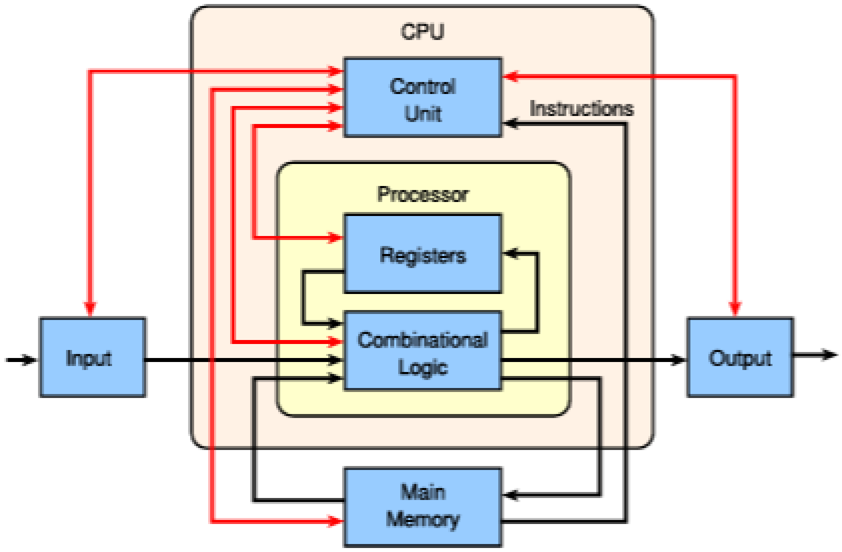
Sign and magnitude

- The MSB is the sign bit (0 = positive, 1 = negative)
- Remaining bits interpreted as magnitude
- Range $[-2^{n-1}-1; 2^{n-1}-1]$
- There are two different zeros: 0 and -0
- A plus -B does not equal A minus B.

Two's Complement method

- Method: invert all bits from 0 to 1 and from 1 to 0 and add 1.
- MSB is a sign bit (0 = positive, 1 = negative)
- Range: $[-2^{n-1}; 2^{n-1}-1]$
- Only one zero
- A plus -B does equal A minus B.
- To convert a number from binary to decimal look at the MSB, is its zero, the number is positive and you just convert it as usual, but if its 1, then take negative 2 to the power of number of bits (digits) and add the number in decimal to it (by converting it the usual way, with the MSB). Example: $1011_b = -(2^4 - 11) = -5.$

CPU Architecture



x86 Architecture

Registers

- x86 has the following 16-bit registers:
    - Four general-purpose ones: `AX`, `BX`, `CX`, `DX`
    - Two address registers: `SI`, `DI`
    - Two pointer registers: `BP`, `SP`
    - One status register: `FLAGS`
    - One instruction pointer: `IP`
- IA-32 has 32-bit versions prefixed with E, so `EAX`, `EBX`, `ECX`, ..., `EIP`
- x86-64 has 64-bit versions prefixed with R, so `RAX`, `RBX`, `RCX`, ..., `RIP`

But even on 32, or 64 bit architectures, lower bit versions of registers also work, so for example to access a 32-bit register on a 64-bit machine you just need to use the right prefix like `EAX`.

Basic Instructions

Instruction	Python equivalent
`MOV dest, src`	`dest = src`
`ADD dest, src`	`dest = dest + src`
`SUB desr, src`	`dest = dest - src`
`MUL src`	`EDX:EAX = EAX * src`
`MUL dest, src`	`dest = dest * src`
`MUL dest, src, const`	`dest = dest * src * constant`
`DIV src`	`EDX, EAX = EDX:EAX / src, EDX:EAX % src`

- In high-level programming languages conditions are checked in `if`, `while`, etc.
- In assembly these operations require a few steps:
    - Performing a check
    - Updating a status register (`EFLAGS` in IA-32)
    - Conditionally jump (or not) to another place in the code
- What does it mean to "jump" in code?
- CPUs have a special *instruction pointer* register
- Normally the CPU:
    - Fetches from the main memory an instruction and its operands
    - Performs the operation
    - Moves the instruction pointer forward to the next instruction
- When jump occurs, the instruction pointer is immediately assigned a new, given value
- The `CMP arg1, arg2` instruction checks the result of `arg1 - arg2` expression and sets flags, e.g.:
    - `SF` (Sign Flag) is 1 if `arg1 - arg2 < 0`
    - `ZF` (Zero Flag) is 1 if `arg1 - arg2 == 0`
- The conditional jumps instructions take note of the flags:

- `JE label`: jump if equal (`ZF == 1`)
- `JNE label`: jump if not equal (`ZF == 0`)
- `JL label`: jump if less (`SF == 1`)
- `JLE label`: jump if less or equal (`SF == 1 OR ZF == 1`)
- `JG label`: jump if greater (`SF == 0`)
- `JGE label`: jump if greater or equal (`SF == 0 or ZF == 1`)