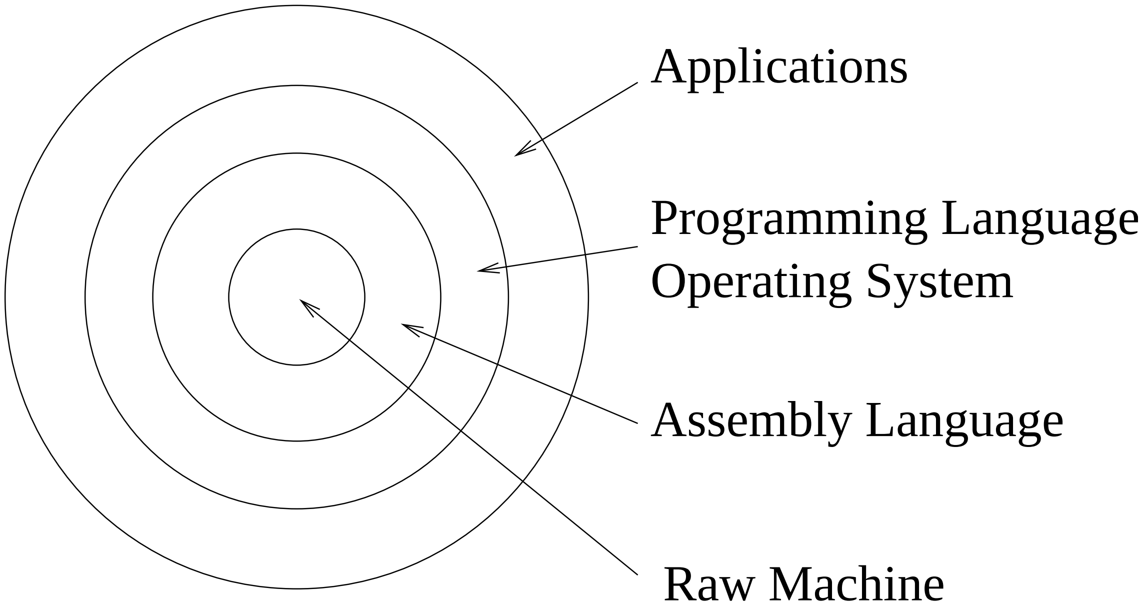
COMP1917: Computing 1

18. Machine Language

Computer Systems

Recall: modern computer systems are layered.



Machine Language Programming

Some useful references for this material:

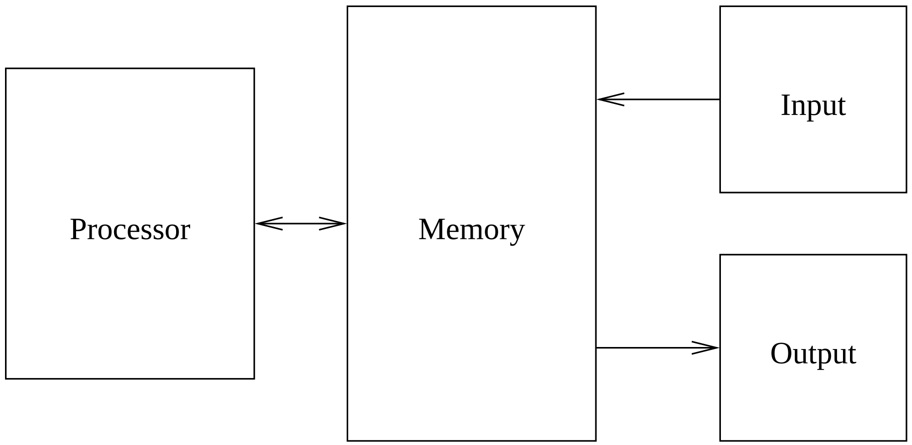
- J. Glenn Brookshear, "Computer Science: An Overview", (4th ed), Benjamin/Cummings, 1994.
- Andrew S. Tanenbaum, "*Structured Computer Organisation*" (3rd ed), Prentice-Hall, 1990.
- David Patterson and John Hennessy, "*Computer Organization and Design: the Hardware/Software Interface*", Morgan-Kauffman, 1994.
- Joseph Byrd and Robert Pettus, "*Microprocessor Systems*", Prentice Hall, 1993.
- Gerry Kane and Joe Heinrich, "*MIPS RISC Architecture*", Prentice Hall, 1992.

History of Computer Technology

	Impact	Hardware Technology	Software Technology
1940's			
	First prototypes		
		Vacuum tubes	
			Machine language
1950's			
	First commercial computers		
			FORTRAN,COBOL,LISP
		Transistors	
1960's			
	Widespread use in business/defence		
			Multi−user Operating Systems
		Integrated circuits	
1970's	Minicomputers		
			Unix and C
		LSI/VLSI	
1980's			
	Microprocessors		
			Windows, Mouse, Menus
		RISC	
			Haskell, Perl, Tcl, VB, OO
1990's			
	Global Network		
			HTML, Java, VRML
⋮			

One constant: the underlying machine model.

Computer Architecture



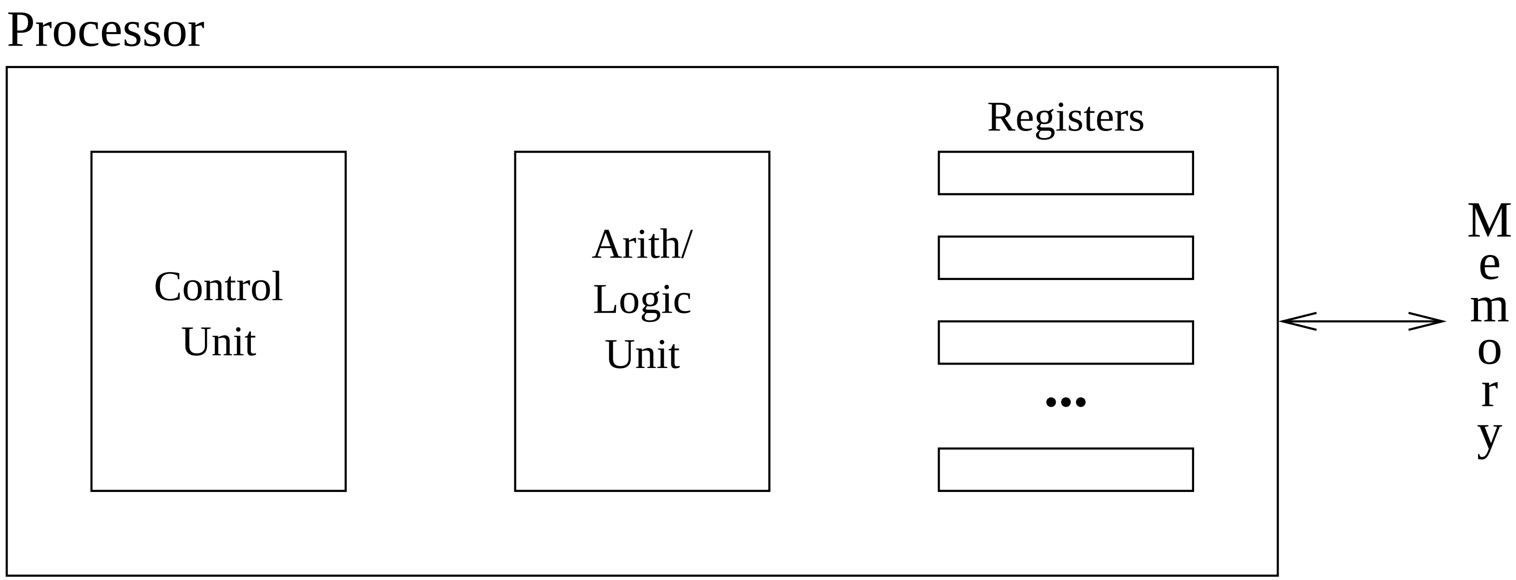
- Processor: control, calculation
- Memory: data & program storage
- Input/output: interface to the world

Input/Output Devices

Vast range of devices are interfaced:

Device	**Read/Write**	**Speed**	**Notes**
disks	r/w	high	high-volume storage
tape	r/w	low	high-volume archiving
cd-rom	r/o	medium	storage
display	w/o	medium	CRT, LC, ...
keyboard	r/o	low	
mouse	r/o	low	1,3-button
other computers	r/w	varying	networks
VR-helmet	r/w	high	games
mechanical	r/w	low	embedded
equipment			systems

Central Processing Unit



Processor (CPU)

The processor's task:

```
{
    Register PC; /* program counter */

    forever {
        fetch instruction from Memory[PC++];
        determine what kind of instruction;
        fetch any necessary data;
        carry out the specified operation;
    }
}
```

Processor Operations

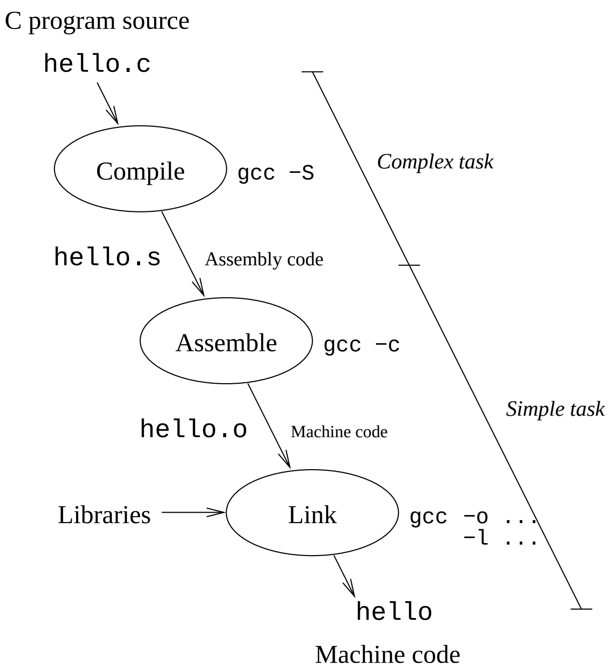
CPUs typically provide operations for:

- data movement (reg-to-reg, reg-to-mem)
- arithmetic calculation (e.g. `+ - * /`)
- logical calculation (e.g. `&& || !`)
- comparison (e.g. `==, >, <, >=, <=`)
- bit manipulation (e.g. `~ & | ^ >> <<`)
- program control (`goto`/`branch`)
- input/output (read, write)

From High-level to Low-level Languages

Real machines can't execute C directly.

Real machines execute their own **machine code**.



Simulated Machine Architecture

We will be using a simulator called `mlsim`, similar to the one described in "Computer Science: An Overview" by J. Glenn Brookshear.

- 256 memory cells, with addresses from `00` to `FF` (hexadecimal) each holding 1 byte (8 bits)
- 16 general-purpose registers (named `R0` to `RF`) each holding 1 byte (8 bits)
- also a 1-byte program counter (`PC`) and a 2-byte instruction register (`IR`).

Machine Language Simulator

```
                          Main Memory
     0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
0   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
1   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
2   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
3   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
4   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
5   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
6   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
7   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
8   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
9   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
A   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
B   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
C   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
D   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
E   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
F   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
R0:00   R1:00   R2:00   R3:00   R4:00   R5:00   R6:00   R7:00   PC: 00
R8:00   R9:00   RA:00   RB:00   RC:00   RD:00   RE:00   RF:00   IR:0000
Type one of the following (H for help):  M, R, P, C, S, G, F, Q:
```

Simulator-Program Commands:

```
Options are as follows:

    M  Change contents of memory cells.
    R  Change contents of registers.
    P  Change contents of program counter.
    C  Clear memory or registers.  (Options will be given.)
    S  Single step. (Execute a single machine cycle.)
    G  Go. (Execute until a halt instruction is executed
            or until 500 machine cycles have been executed.)
    F  List file options (to save or retrieve programs).
    H  Display this help screen.
    Q  Terminate this simulation program.
```


Machine Language Instructions

- each machine instruction is 2 bytes (16 bits) long
- the first 4 bits comprise the op-code
- the remaining 12 bits make up the arguments or “operands”

Machine Language Instructions

Op	Args	Description
1	RXY	LOAD register R with contents of memory cell whose address is XY.
2	RXY	LOAD register R with the bit pattern XY.
3	RXY	STORE the contents of register R in the memory cell with address XY.
4	0RS	COPY the bit pattern found in register R to register S.
5	RST	ADD the bit patterns in registers S & T as though they are 2's complement integers, and store the result in register R.
6	RST	ADD the bit patterns in registers S & T as though they are floating point numbers, and store the floating point result in register R.
7	RST	OR the bit patterns in registers S & T and store result in register R.
8	RST	AND the bit patterns in registers S & T and store result in register R.
9	RST	XOR the bit patterns in registers S & T and store in register R.
A	R0X	ROTATE the bit pattern in register R one bit to the right X times.
B	RXY	BRANCH to the instruction located in the memory cell at the address XY if the bit pattern in register R is equal to the bit pattern in register 0; otherwise, continue with the normal sequence of execution.
C	000	HALT execution.

Machine Language Examples

14A3	store contents of the memory cell at address A3 into register R4.
20A3	store the value A3 into register R0.
35B1	store the contents of register R5 into memory cell at address B1.
40A4	copy the contents of register RA (R10) into register R4.
5726	add binary values in registers R2 & R6 and store the sum in register R7.
634E	add the bit values in registers R4 and RE (R14) as floating-point numbers and store the result in register R3.
7CB4	OR the contents of RB (R11) & R4 and store the result in register RC (R12).
8045	AND the contents of registers R4 and R5 and store the result in register R0.
95F3	XOR the contents of RF (R15) & R3 and store the result in register R5.
A403	rotate the contents of register R4 3 bits to the right in a circular fashion.
B43C	compare the contents of register R4 with the contents of register R0; if the two are equal, pass control to the instruction at memory address `3C`; otherwise, continue execution in its normal sequence.
C000	halt execution.

AND Truth Table

- AND — logical and
- C operator: `&`

x	y	x & y
0	0	0
0	1	0
1	0	0
1	1	1

OR Truth Table

- OR — logical inclusive or
- C operator: `|`

x	y	x | y
0	0	0
0	1	1
1	0	1
1	1	1

XOR Truth Table

- XOR — logical exclusive or
- C operator: `^`

x	y	x ^ y
0	0	0
0	1	1
1	0	1
1	1	0

Machine Language Programming

For this simple machine each complete instruction is a 4-digit (hexadecimal) number, stored across two memory locations.

Why don't we program in Machine Language?

Here is an example of a simple machine-language program, starting at address `10`:

```
234B121C5023302AC0003A
```

To make any sense out of this at all, we need to group the numbers into pairs, and add extensive in-line comments...

Machine Language Programming

Address	Content	Description
10	234B	; Load (hex) value 4B into R3
12	121C	; Load value at loc. 1C into R2
14	5023	; Add R2, R3 as integers, put result into R0
16	302A	; Store value in R0 into loc. 2A
18	C000	; Halt
. .	. .	
1C	3A	; Data value

If:

- `m` denotes value in address `1C`
- `n` denotes value in address `2A`

then the C code corresponding to the above program (using decimal equivalents of hex. numbers):

```
int m = 58, n;
n = m + 75;
```

Addressing Modes

This program illustrates the difference between immediate and direct addressing.

- The first instruction `234B` simply loads the specified argument `4B` into `R3` (immediate addressing).
- The second instruction `121C` goes to memory location `1C`, fetches whatever is stored there, and loads it into `R2` (direct addressing)
- Some machines also allow indirect addressing, where the machine goes to the specified memory location to get an address, then fetches the value at that address.

Negation

How do we compute the negative of a number?

Step 1: XOR with FF (1's complement)

Step 2: ADD 1 (2's complement)

Note: for floating-point numbers, we only need to change the first bit, by XOR-ing with 80.

Subtraction

Using the instruction set for this simple machine, write a machine-language routine equivalent to the C code:

```
int m;
int n;
int k;

k = m - n;
```

Assume that `m`, `n` and `k` are stored in memory locations `10`, `12` and `14`, respectively, and that program execution begins from location `20`.

Subtraction Program

```
Address  Contents  Description
  10        38      ;  Data   m
  12        15      ;  Data   n
  14        00      ;  Result k
  ..        ..
  20       1310     ;  LOAD  val at loc.10 to R3 // m
  22       1412     ;  LOAD  val at loc.12 to R4 // n
  24       2101     ;  LOAD  01 to R1
  26       22FF     ;  LOAD  FF to R2
  28       9524     ;   XOR  R2,R4; result to R5
  2A       5515     ;   ADD  R1,R5; result to R5
  2C       5535     ;   ADD  R3,R5; result to R5
  2E       3514     ;  STORE R5 in loc.14        // k
  30       C000     ;  HALT
```


Multiplication

Write the machine-language coding for this model machine that is equivalent to the C statements:

```
int m = 12, n;
n = 3 * m;
```

Alternative form: This machine has only one arithmetic operation – ADD (in 2 forms). So we first convert the C code to:

```
int m = 12, n=0, i=0;

while ( i < 3 ) {
   n = n + m;
   i = i + 1;
}
```


Multiplication

Allocate memory and registers:

- Loc. 10 for variable `m`
- Loc. 12 for variable `n`
- R0 for constant 3
- R1 for constant 1
- R2 for counter `i`
- R3 for value of `m`
- R4 for accumulating value of `n`

Multiplication Program

```
Address   Contents   Description
  10         12       ;  Data    m
  12         00       ;  Result n

  20        1310      ;  Load  val at loc.10 to R3 // m
  22        2003      ;  Load  03 to R0 (multiplier)
  24        2101      ;  Load  01 to R1 (incrementer)
  26        2200      ;  Load  00 to R2            // i = 0
  28        2400      ;  Load  00 to R4            // n = 0
  2A        B232      ;  If   (R2 == R0) branch PC to 32
  2C        5443      ;  Add   R4,R3; result to R4 // n = n + m
  2E        5221      ;  Add   R2,R1; result to R2 // i = i + 1
  30        B02A      ;  Jump  PC to 2A
  32        3412      ;  Store R4 in loc.12        // n
  34        C000      ;  Halt
```

Exercise

Some variations:

What changes are needed to code:

- `n = a * m;`
- `n = m ^ a;`

Other Bit-wise operations

How do we compute “sign of a number”?

AND with 80

the result is:

80, if $n < 0$

00, if $n \geq 0$

What happens if an ASCII lower-case letter is AND-ed with DF ?

What happens if an ASCII upper-case letter is OR-ed with 20 ?

Exercise

Using the machine-language instruction set for this model machine, describe the machine operations required to execute the following fragment of C coding:

```
int a, b, min;

min = b;
if( a < b ) {
   min = a;
}
```