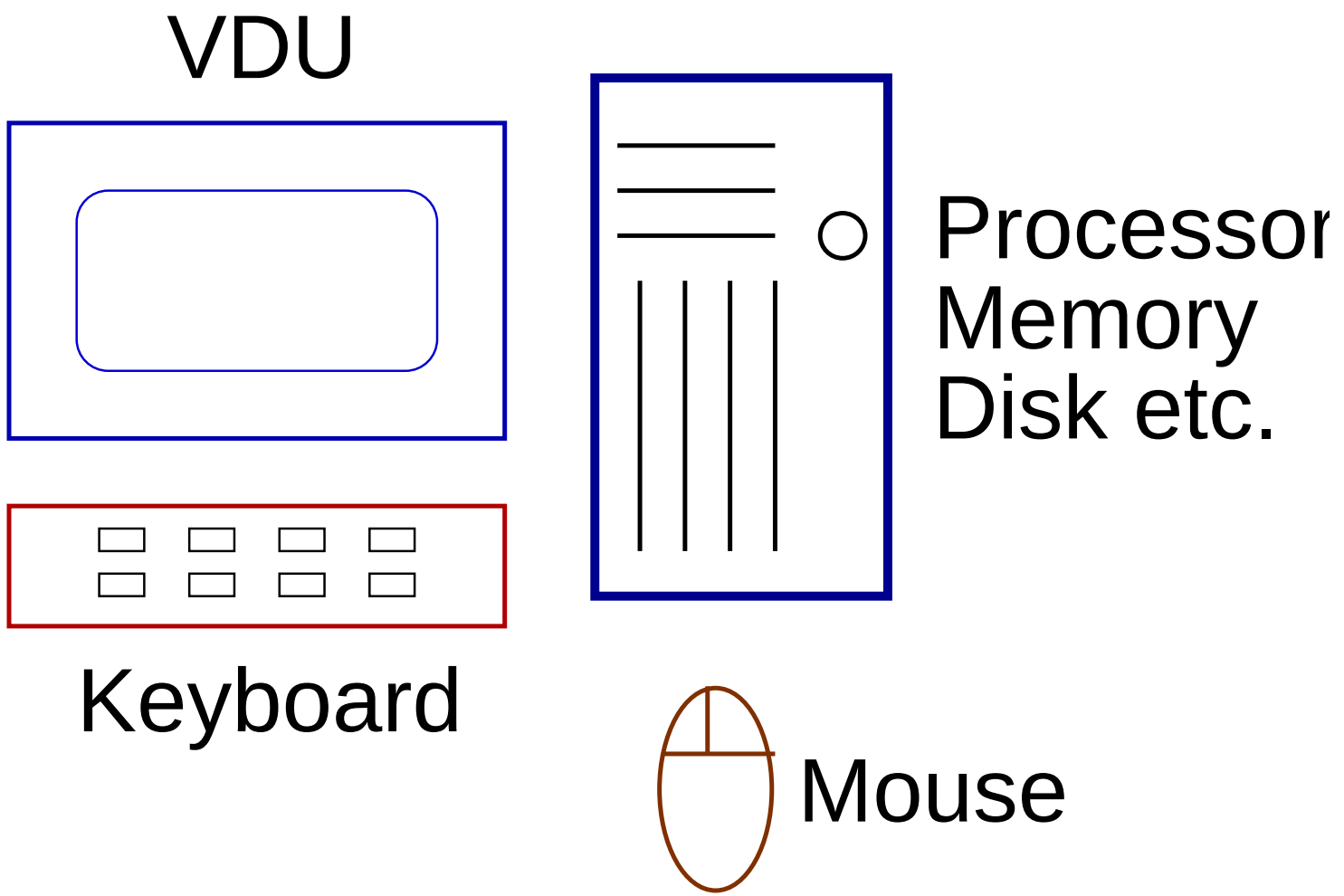
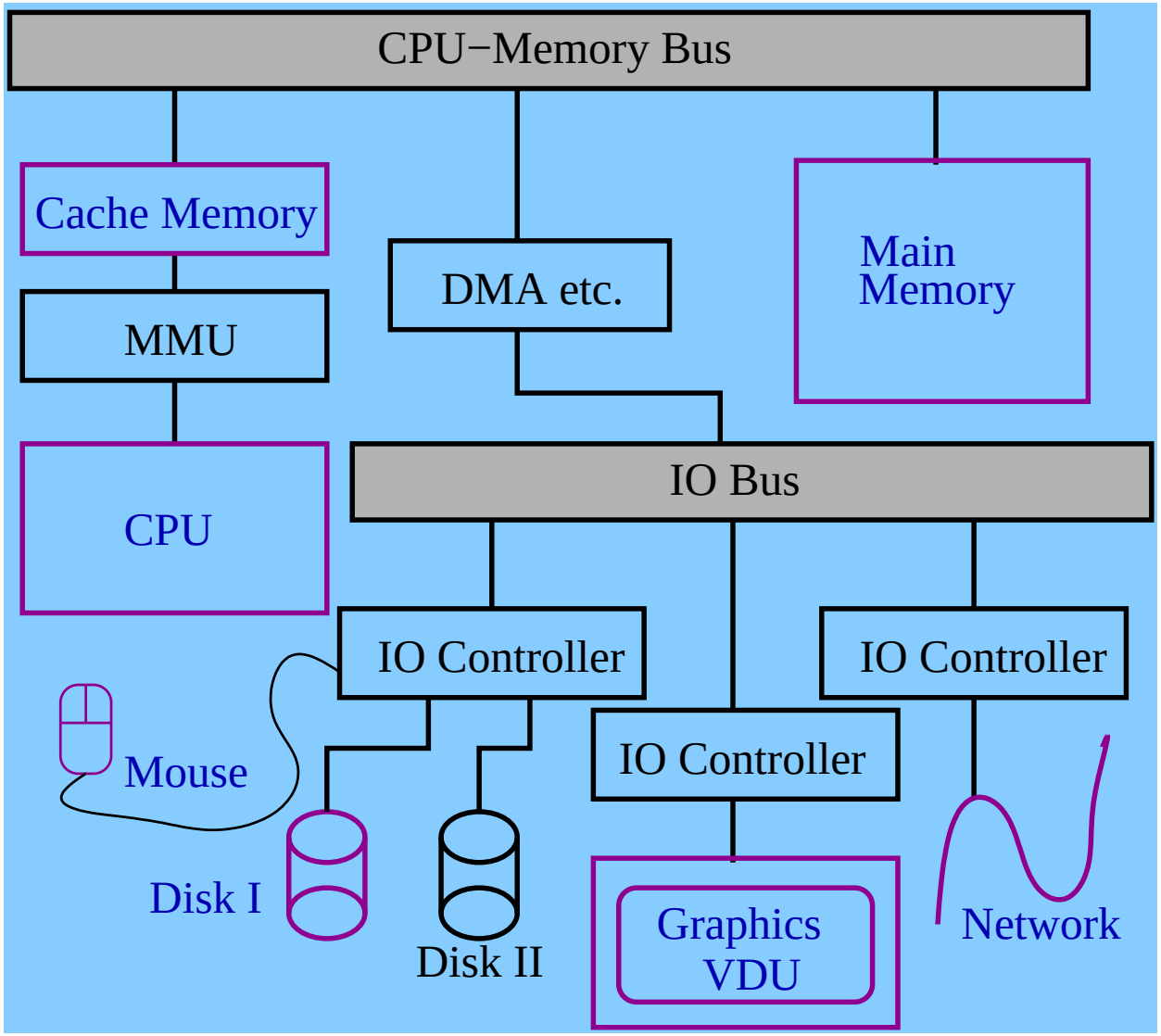
Computer System: *An Overview*


VDU
Processor
Memory
Disk etc.
Keyboard
Mouse


CPU−Memory Bus
Cache Memory
MMU
CPU
DMA etc.
Main Memory
IO Bus
IO Controller
IO Controller
IO Controller
Mouse
Disk I
Disk II
Graphics VDU
Network

Stored Program Computer

- A stored program computer is used to process data.
- The CPU of a computer follows a stream of instructions.
- A finite sequence of instructions, called a program, is used to process data.
- Both program and data are stored in the computer memory.

von Neumann Computer

- A computer model where both data and program are stored in the same memory is called von Neumann architecture[a].
- Data and program are stored in different memory in the Harvard architecture[b].

[a] The first draft of a report on the EDVAC at the he Moore School of Electrical Engineering, by John von Neumann, and the design of ENIAC by J. P. Eckert and John W. Mauchly at the University of Pennsylvania

[b] Harvard Mark I electro mechanical computer stored instructions on punched tape and data in electro-mechanical counters.

CPU Instruction Set

- A finite set of (machine) instructions is associated with every CPU. This is called the instruction set of the CPU.
- A program for a particular computer finally should consist of a sequence of instructions of the corresponding CPU.
- Each machine instruction is a finite length string of binary digits (bits).

CPU Instruction Set

- Instruction set for different types of CPUs e.g. Pentium, PowerPC, SPARC, x86-64 etc. are different.
- Instruction ment for one CPU is not understandable by another CPU. So the machine language program of one computer need not run directly on another machine.

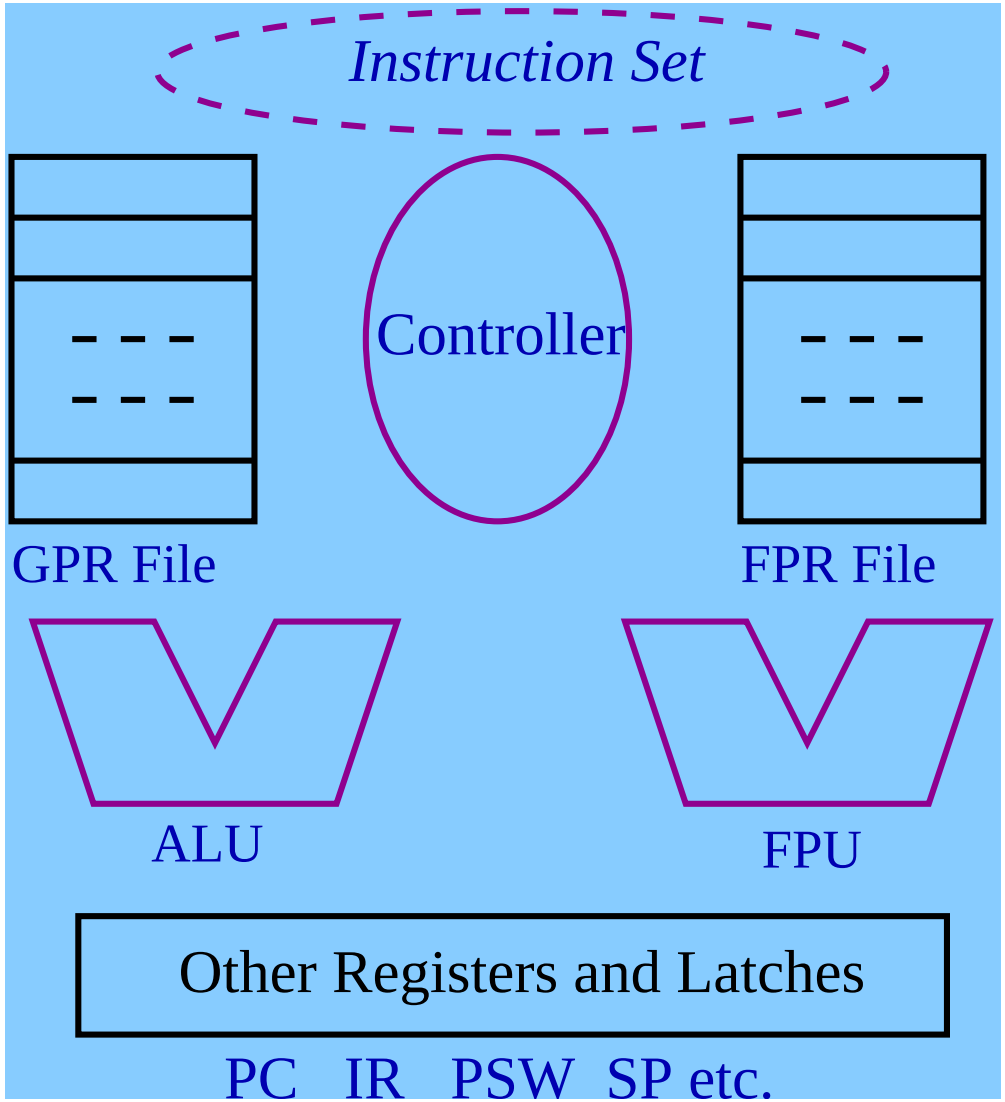
Fetch-Execute Cycle

- (Fetch) The CPU fetches the next instruction from the memory[a] and decodes it.
- (Execute) Depending on the nature of the instruction, the CPU may fetch the required data from the memory[b], and process it.
- This cycle continues.

[a] Main memory where the program in execution is stored.

[b] Main memory where the data is stored.

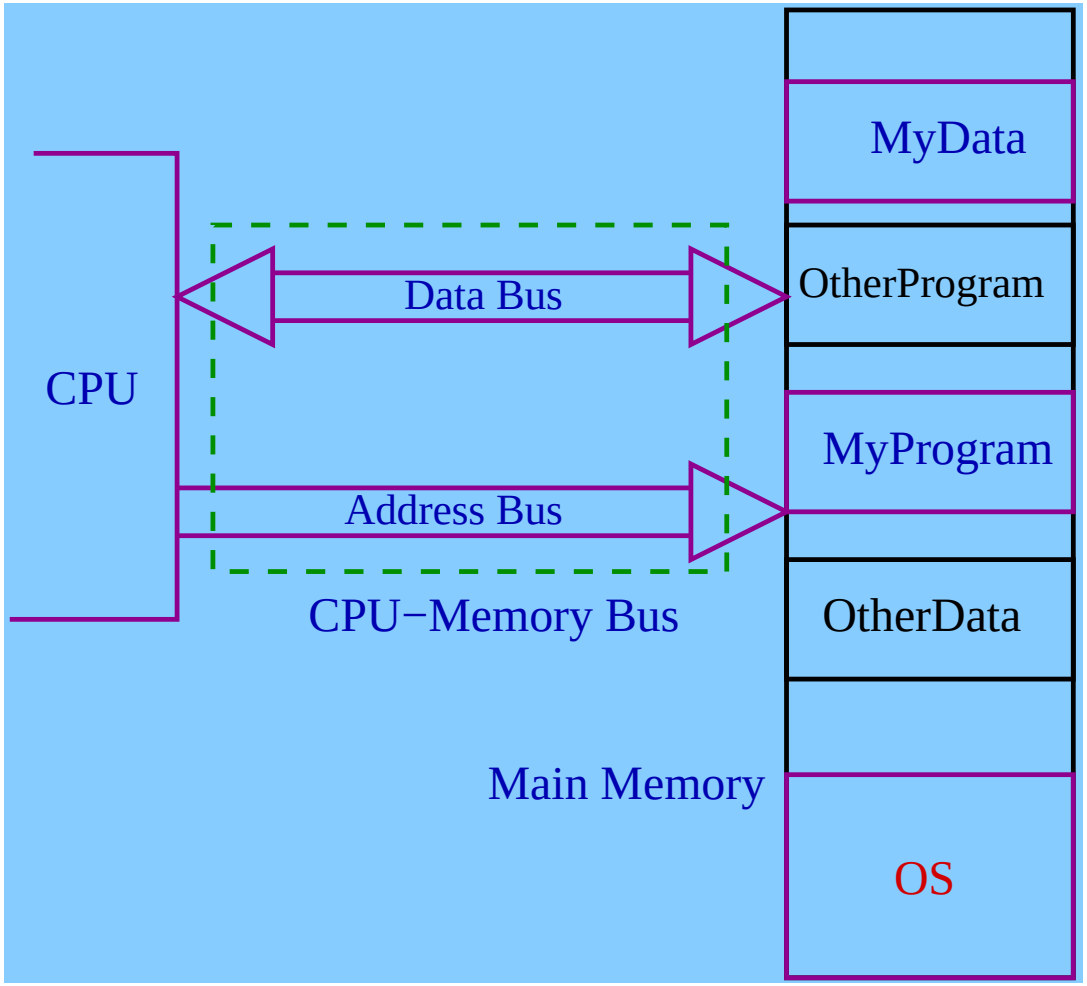
Central Processing Unit (CPU)


Instruction Set
Controller
GPR File
FPR File
ALU
FPU
Other Registers and Latches
PC IR PSW SP etc.

Central Processing Unit (CPU)

- GPRs: general purpose registers
- FPRs: floating-point registers
- PC or IP: program counter or instruction pointer
- PSW: program status word
- IR: instruction register

Program and Data in the Main Memory



Memory Location

- Main memory is divided into equal size blocks[a] called memory location.
- Each memory location has a unique address. The location can be activated by sending its address to the memory subsystem.

[a] Typical size of each block is 1, 2, 4, or 8 bytes. One byte consists of eight binary digit (8-bit).

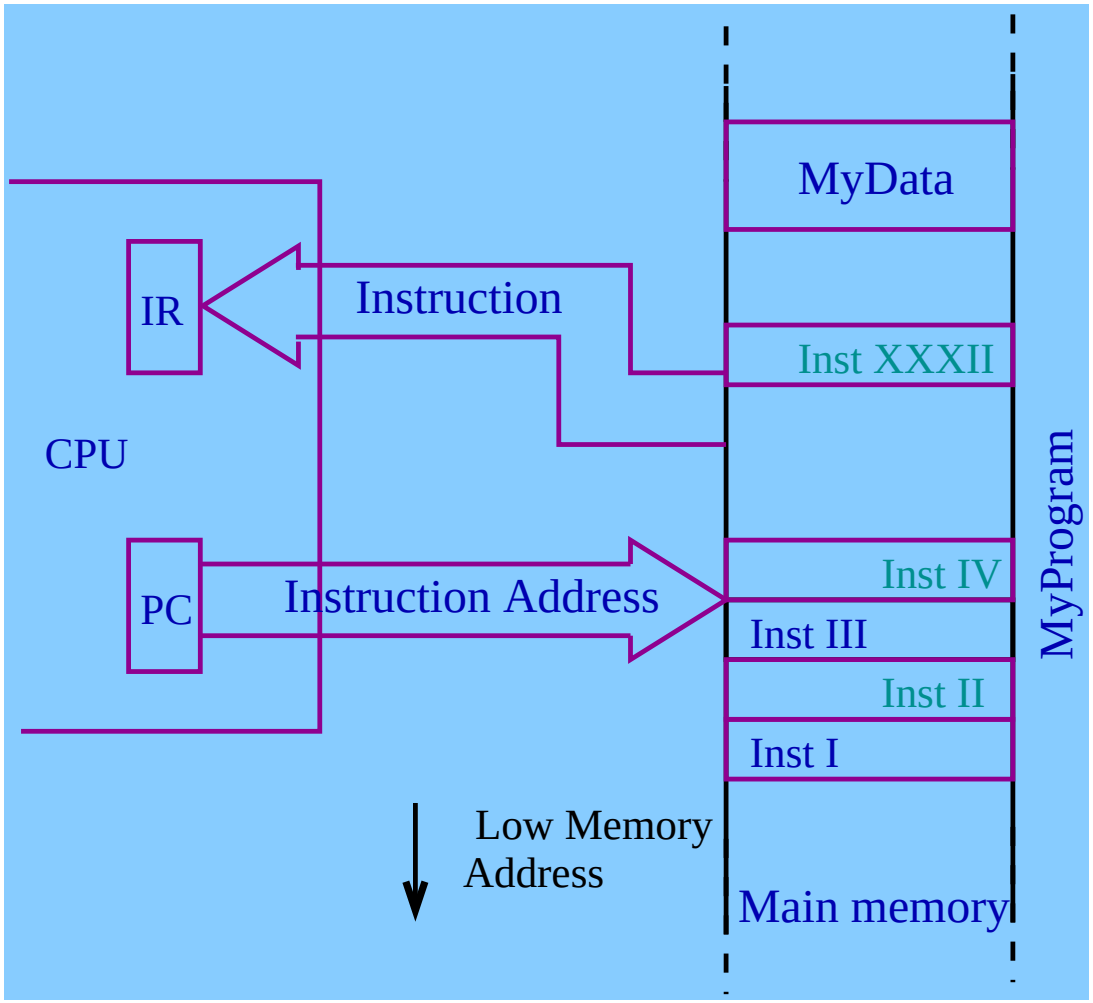
Program Execution

Instruction Fetch

- The program counter or the instruction pointer (PC or IP) holds the address of the next instruction in the main memory.
- The CPU sends the instruction address on the address bus.

Instruction Fetch

- The memory subsystem reads the addressed location and sends the instruction on the data bus.
- The CPU saves the instruction in the instruction register (IR).


MyData
IR
Instruction
Inst XXXII
CPU
PC
Instruction Address
Inst IV
Inst III
Inst II
Inst I
MyProgram
Low Memory
Address
Main memory

Instruction Decoding

The instruction is decoded by the CPU hardware (or firmware) to generate the sequence of following actions by the CPU.

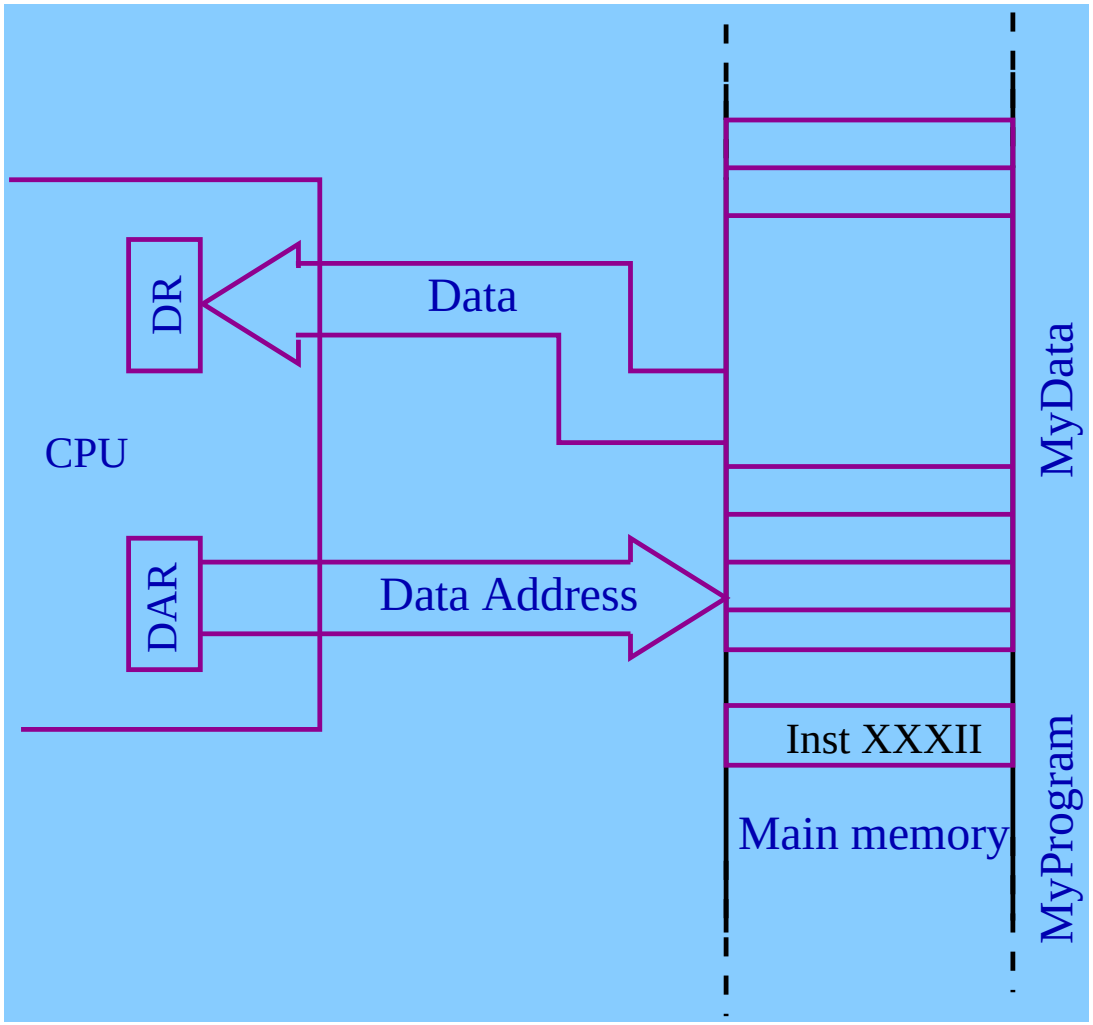
Data Fetch

- If the processing of the instruction requires data from the memory, the CPU fetches it by sending the data address[a] on the address bus.
- Memory subsystem dispatches the data on the data bus.

[a] Data address is already available in the CPU either as a part of the instruction (IR) or in some other CPU register.

Data Fetch

The CPU may save the data in one of its internal registers and use it for the required operation.


DR
Data
CPU
DAR
Data Address
Inst XXXII
Main memory
MyData
MyProgram

Data Write

- The result of the operation may be stored in an internal register of the CPU or in the memory.
- The CPU sends the address of the memory location and the data to write, on the address and the data buses respectively, along with the memory-write signal.

With a Grain of Salt

The actual process is more complex

A Machine Instruction of Pentium

1000 0011 1110 1100 0000 1000

Execution of this instruction by the CPU subtracts eight (8) from the content of an internal register called stack-pointer (`esp`). It is difficult to make head and tail out of the binary string representing an instruction.

`esp ← esp - 8`

Assembly Language Instruction

The first step is to provide a more human understandable symbolic representation of the machine instructions. These are called assembly language instructions.

1000 0011 1110 1100 0000 1000 ⇒ sub 8, esp

Assembly language Program

- A sequence of assembly language instructions forms an assembly language program.
- Assembly language depends on the type of the CPU and also on the names etc. given to the instructions.

Assembly Language to Machine Language

- The CPU does not understand assembly language instruction.
- A program (system software) called assembler translates an assembly language program to the machine language program.
- Input to an assembler is an assembly language program and its output is the equivalent machine language program.

High-Level Languages

- It is also tedious to write big application program in assembly language.
- Moreover an assembly language program depends on a particular CPU and also on the assembler.

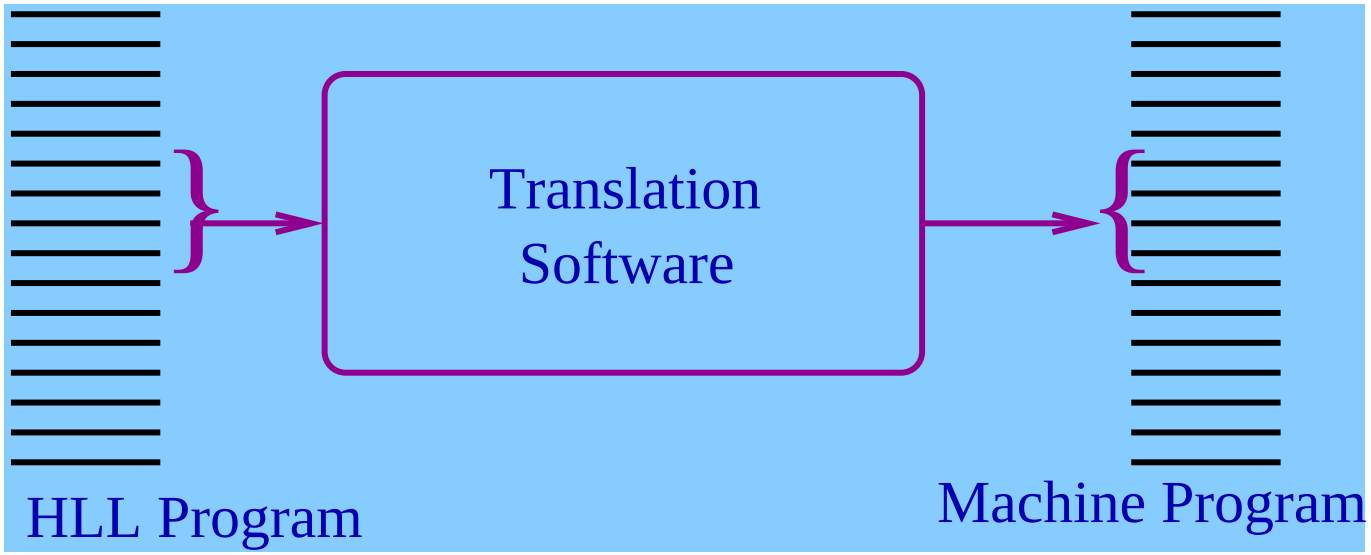
High-Level Languages

- People designed more human understandable programming languages by introducing mathematical symbols and more abstractions. These are called high-level languages.
- High-level programming languages are suppose to be machine independent.

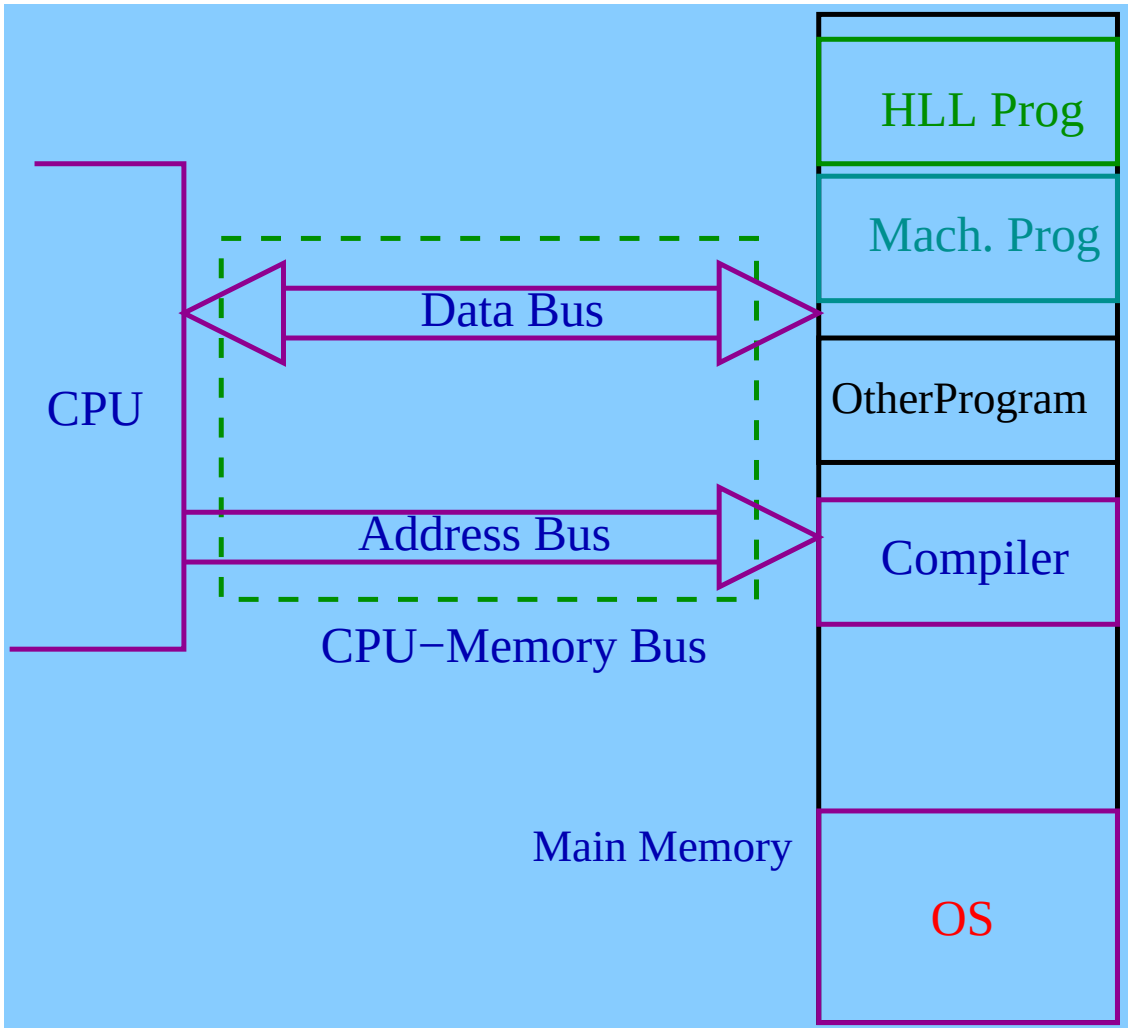
Translation

- Programs in high-level languages cannot be used to control the CPU directly.
- Softwares to translate programs from a high-level language to some assembly language or machine language is necessary.
- Such a system software is called a compiler or an interpreter.

Compiler: *A Software Translator*



Compiler: *A Software Translator(cont.)*

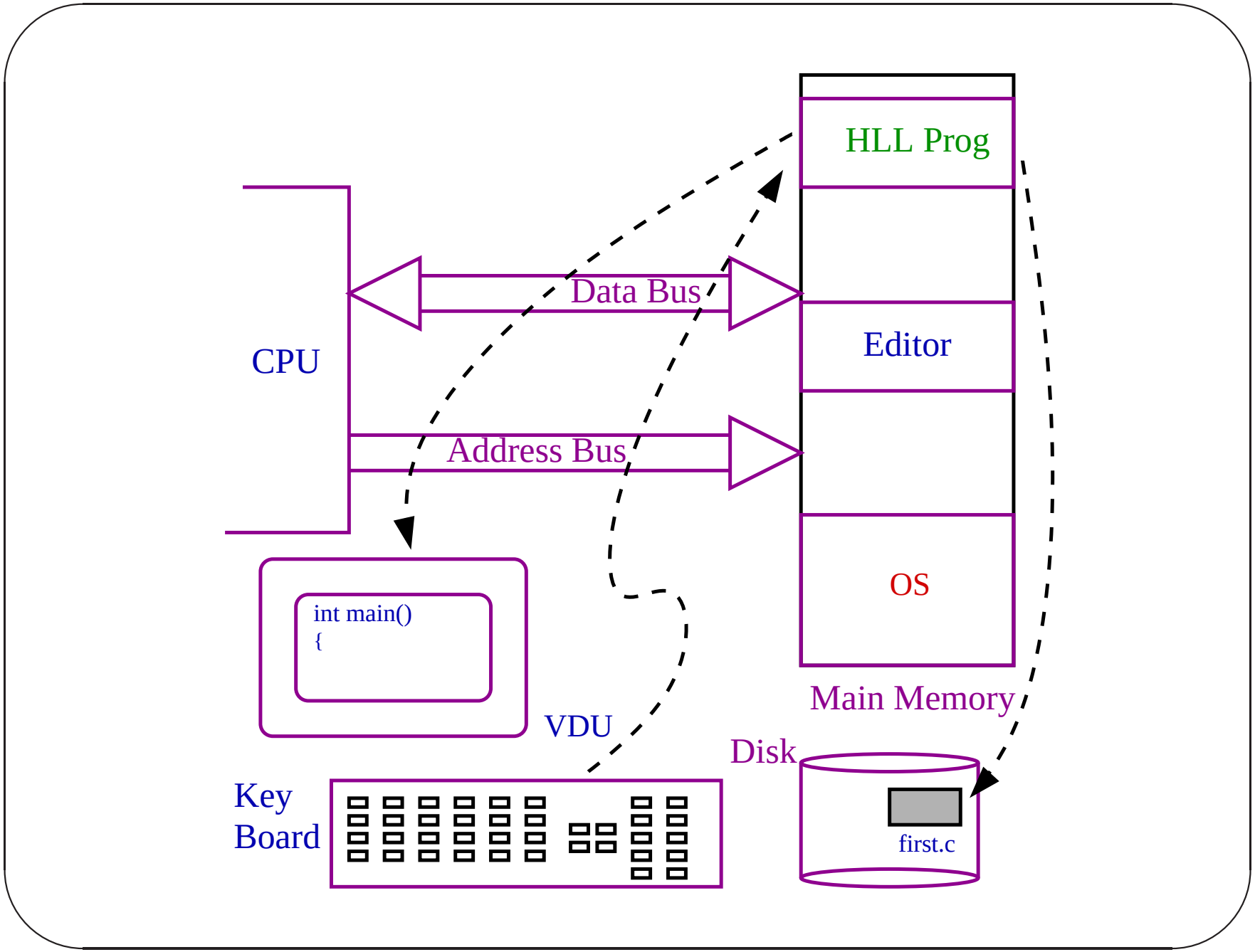


Writing and Storing a HLL Program

It is necessary to get some facility to write and store a high-level language or assembly language program in a computer. This is provided by a software called an editor e.g. vi, emacs etc.

Writing and Storing a HLL Program

These softwares provide facility to edit a text and store it in the hard disk as a named object called a file.


HLL Prog
Data Bus
CPU
Editor
Address Bus
OS
int main()
{
Main Memory
VDU
Disk
Key
Board
first.c

Operating System

A computer system is very difficult to use unless a core master program called an operating system (OS) is running on it to make it user friendly. It provides a better view of the available resources and also manages them.

Command Interpreter

There are other system softwares (utilities) that are essential for the ease of use. One most important is the command interpreter e.g. bash. We shall talk more about it afterward.

Our programming Environment

- PC with Intel Core 2 Duo CPU(+ MMU + cache) (Intel Xeon quad code x86-64 in the Lab.).
- Operating System, Linux (SunOS in the Lab.).
- Command interpreter, bash (Bourne-again shell).
- Editor, emacs or vi.

Our programming Environment

- Compiler, GNU gcc for C language[a].
- Assembler, as.

[a] gcc is actually a C++ compiler.

Programming Environment in Lab

- Thin Client and
- Server

Thin Client

A thin client is a computer (and necessary software) that does most its computational job on a more powerful server. Large number of clients share the same server.
A modern thin client is a computer terminal that provides graphical interface to the user. But for other functionality e.g. the OS, it uses the server.
This is a modern version of VDU terminal and main-frame computer connected over network.

Thin Client

- Sun Ray Display Client -
- *Display*: 17-inch, LCD, flat-panel,
- *Graphics*: Integrated 24bit, 2D graphics,
- *I/O*: USB mouse, keyboard, smart-card reader (ISO-7816-1), audio (CD-quality audio in/out), microphone,
- *Network*: 10/100 BASE-T connection to LAN,
- *Ports*: USB, serial, audio, video.

The Server

- Sun Fire X4170 Server
  - Intel Xeon 5570, 4-Core, 8-Thread,
  - 64-bit AMD64 (x86-64) architecture,
  - Clock 2.93 GHz,
  - *Cache*:

    L1: 32 KB (I) + 32 KB (D) per core,

    L2: 256 KB per core,

    L3: 8 MB per processor,

- *Main Memory* : **144 GB** (Max),
- *Hard Disk* : 2× 146 GB.

OS, Editor and Compiler

- *OS* : **SunOS**
- *Programming Language* : **C**
- *Compiler* : **gcc** (**cc**)
- *Editor* : **emacs**
- *Command-interpreter*: **bash**

First C Program

```
#include <stdio.h>
int main()
{
    printf("The First C Program\n");
    return 0;
} // first.c
```

Write, Compile and Execute on Linux OS

1. Use an editor e.g. 'vi' or 'emacs'.
   - Run (execute) the editor program.
   - Key-in the C program text.
   - Save the program as a named file e.g. 'first.c'.
2. Compile the C program to the executable file '`a.out`'.

3. If there is an error, go back to the editor and fix it; otherwise run the '`a.out`' file and get the output.

Typical Commands Are

- `$ emacs first.c &`
- `$ cc -Wall first.c`
- `$ ./a.out`

  `My first program`

File Type

- `$ file first.c`

  `first.c: ASCII C program text`

- `$ file a.out`

  `a.out: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5, dynamically linked (uses shared libs), not stripped`

Note

- The compiler creates the executable file[a] 'a.out' (assembler output) from the C program file 'first.c'.

[a] It contains the machine code and some other data structure.

The Second Program

```
#include <stdio.h>
#define MAX 10
int main()
{
    int n;

    printf("Enter the Data: ");
    scanf("%d", &n);
    if(n>MAX) printf("\nThe %d > %d\n", n, MAX);
    else printf("\nThe %d <= %d\n", n, MAX);
    return 0;
} // second.c
```

In the laboratory

```
$ bash
bash-3.00$ gcc -Wall first.c
bash-3.00$ ./a.out
The First C Program
```

This Part Need not be Covered

An Assembly Language Program

```
// Compile as follows
//      $ /lib/cpp first.S first.s
//      $ as -o first.o first.s
//      $ ld first.o
// You may also use $ cc first.S
// but in that case change _start by main

#include <asm/unistd.h>
#include <syscall.h>
```

```
#define STDOUT_FILENO 1


.file "first.S"
.section        .rodata
L1:
    .string "My first program\n"
L2:


.text
.globl _start
```

```
_start:
        movl   $(SYS_write), %eax
        movl   $(STDOUT_FILENO), %ebx
        movl   $L1, %ecx
        movl   $(L2-L1), %edx
        int    $128

        movl   $(SYS_exit), %eax
        movl   $0, %ebx
        int    $128
        ret
```

Assemble and Execute

- `$ /lib/cpp first.S first.s`
- `$ as -o first.o first.s`
- `$ ld first.o`
- `$ /a.out`

  `My first program`