System Programming

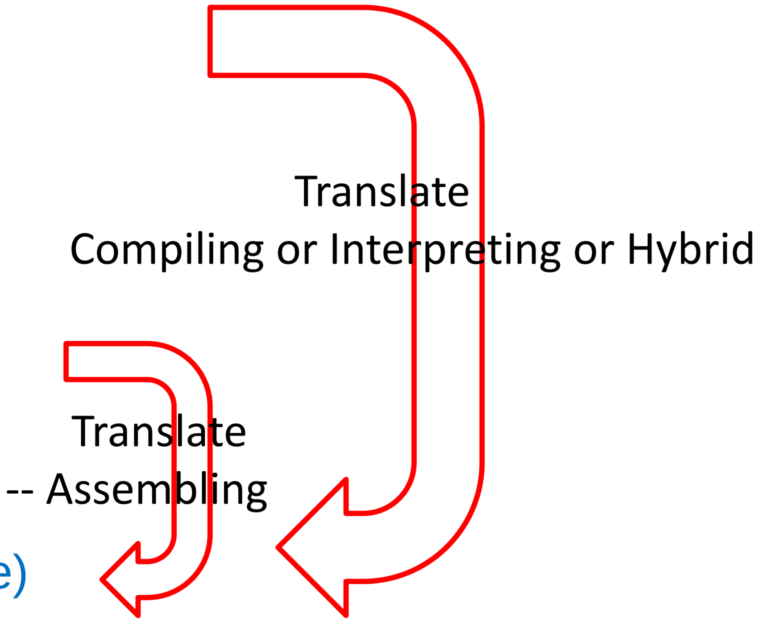
- Design and Implementation of system software.
- System Software: a variety of programs supporting the operation of a computer.
- Typical system programs: OS, Complier, Assembler (Linker, Loader, Macro Processors), Text Editor, Debugger, …,
  - Their functions and relations among them?
- Why specific and important:
  - rely on hardware (computer architecture)
  - need related low level languages such as assembly language
  - Free users from knowing the detail of machines.

Relations?

- Programming Languages:
  - Machine Language (ML)→ Assembly Language (AL)→ High-Level Programming Language (HL)
    - ML: machine code, i.e, binary code, e.g., 01100110
    - AL: mnemonic instructions, e.g., STO (Store)
    - HL: statements, e.g. if … Then … else …
  - Programs in HL → Object Code (in AL or ML)
    - (Compiler)
  - Programs in AL → Object Code (in ML)
    - (Assembler)
- Four typical systems
  - Operating System, Compiling System, Database (Administration) System, Network System

4 Level Programming languages

- 4th generation language
  - Specify just what?
- High level language (HL):
  - Statements
  - Specify what and how?
- Assembly language (AL):
  - Mnemonic instructions
- Machine language (ML):
  - 0/1 instructions (Executable)



C code, AL code, and ML code of the same program

```c
#include <stdio.h>
int main(){
    int a,b,c,sumMul;
    printf("Please input three integers: ");
    scanf("%d %d %d",&a,&b,&c);
    sumMul=(a+b)*c;
    printf("\n(%d+%d)*%d =  %d.\n",a,b,c,sumMul);
    return 0;
}
```

```
push    %rbp
mov     %rsp,%rbp
sub     $0x10,%rsp
mov     $0x400690,%edi
mov     $0x0,%eax
callq   0x400460 <printf@plt>
lea     -0x10(%rbp),%rcx
lea     -0xc(%rbp),%rdx
lea     -0x8(%rbp),%rax
mov     %rax,%rsi
mov     $0x4006ae,%edi
mov     $0x0,%eax
callq   0x400480 <__isoc99_scanf@plt>
mov     -0x8(%rbp),%edx
mov     -0xc(%rbp),%eax
add     %eax,%edx
mov     -0x10(%rbp),%eax
imul    %edx,%eax
mov     %eax,-0x4(%rbp)
mov     -0x10(%rbp),%ecx
mov     -0xc(%rbp),%edx
mov     -0x8(%rbp),%eax
mov     -0x4(%rbp),%esi
mov     %esi,%r8d
mov     %eax,%esi
mov     $0x4006b7,%edi
mov     $0x0,%eax
callq   0x400460 <printf@plt>
mov     $0x0,%eax
eaveq
retq
```

Machine code (Executable) in binary

```
11100101100010010100100001010101    00010000111011001000001101001000
01000000000001101001000010111111    00000000000000001011100000000000

10111100111010000000000000000000    01001000111111111111111111111110
01001000111100000100110110001101    01001000111010001010101100011101

01001000111110000100010110001101    10101110101111111100011010001001
10111000000000001000000000000110    00000000000000000000000000000000

11111111111111010111111011101000    11111000010101011000101111111111
00000001111101000100010110001011    11110000010001011000101111000010

00010011100001010101111100001111    01001101100010111111110001000101
11110100010101011000101111110000    10001011111110000100010110001011

10001001010000011111110001110101    10111111110001101000100111110000
00000000100000000000011010110111    00000000000000000000000010111000

11111110011011011110100000000000    00000000101110001111111111111111
11001001000000000000000000000000    00011111000011110110011011000011

01000001000000000000000001000100    11111111100010010100000101010111
10001001010010010101011001000001    01001001010101010100000111110110
```

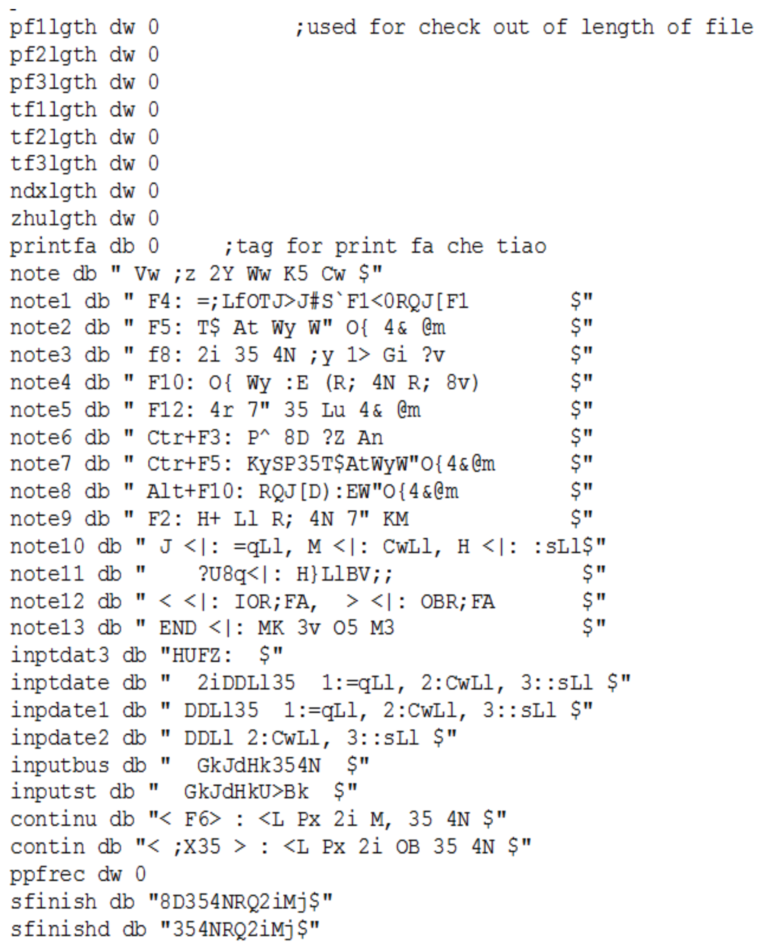
The machine is:

Dell PowerEdge R430 Server

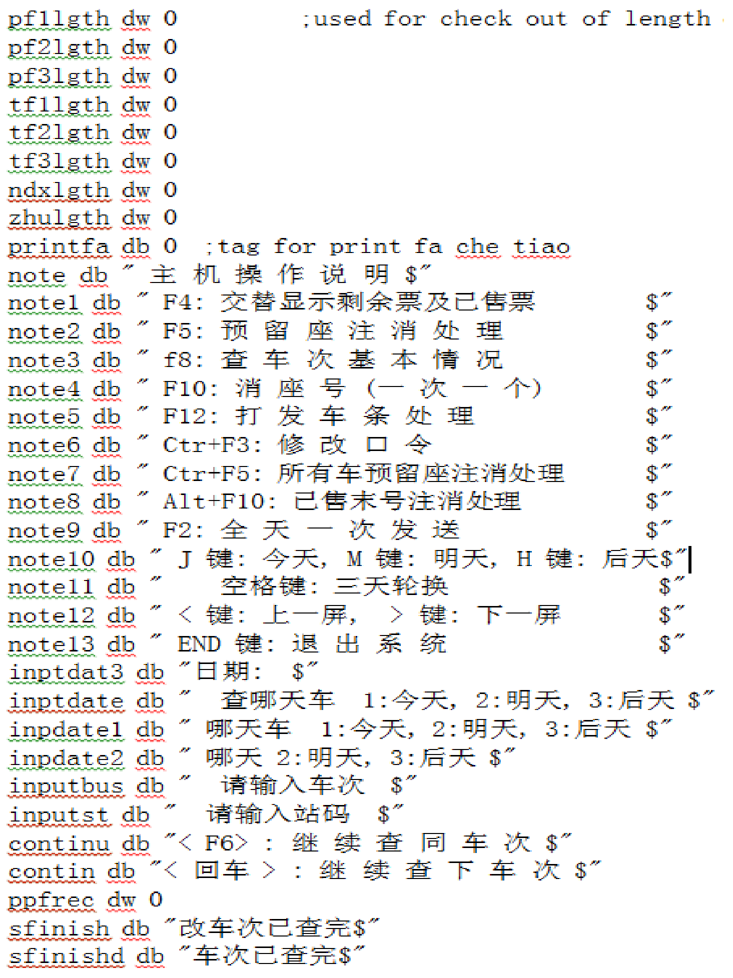
The CPU is:

Intel Xeon E5-2640 v3 2.6GHz

A real commercial system
written in Intel 80286 CPU Assembly language



```
pf1lgth dw 0          ;used for check out of length of file
pf2lgth dw 0
pf3lgth dw 0
tf1lgth dw 0
tf2lgth dw 0
tf3lgth dw 0
ndxlgth dw 0
zhulgth dw 0
printfa db 0     ;tag for print fa che tiao
note db " Vw ;z 2Y Ww K5 Cw $"
note1 db " F4: =;LfOTJ>J#S`F1<0RQJ[F1        $"
note2 db " F5: T$ At Wy W" O{ 4& @m          $"
note3 db " f8: 2i 35 4N ;y 1> Gi ?v          $"
note4 db " F10: O{ Wy :E (R; 4N R; 8v)       $"
note5 db " F12: 4r 7" 35 Lu 4& @m            $"
note6 db " Ctr+F3: P^ 8D ?Z An               $"
note7 db " Ctr+F5: KySP35T$AtWyW"O{4&@m      $"
note8 db " Alt+F10: RQJ[D):EW"O{4&@m         $"
note9 db " F2: H+ Ll R; 4N 7" KM             $"
note10 db " J <|: =qLl, M <|: CwLl, H <|: :sLl$"
note11 db "    ?U8q<|: H}LlBV;;              $"
note12 db " < <|: IOR;FA,  > <|: OBR;FA      $"
note13 db " END <|: MK 3v O5 M3              $"
inptdat3 db "HUFZ:  $"
inptdate db "  2iDDLl35  1:=qLl, 2:CwLl, 3::sLl $"
inpdate1 db " DDLl35  1:=qLl, 2:CwLl, 3::sLl $"
inpdate2 db " DDLl 2:CwLl, 3::sLl $"
inputbus db "  GkJdHk354N  $"
inputst db "  GkJdHkU>Bk  $"
continu db "< F6> : <L Px 2i M, 35 4N $"
contin db "< ;X35 > : <L Px 2i OB 35 4N $"
ppfrec dw 0
sfinish db "8D354NRQ2iMj$"
sfinishd db "354NRQ2iMj$"
```



```
pf1lgth dw 0          ;used for check out of length
pf2lgth dw 0
pf3lgth dw 0
tf1lgth dw 0
tf2lgth dw 0
tf3lgth dw 0
ndxlgth dw 0
zhulgth dw 0
printfa db 0  ;tag for print fa che tiao
note db ” 主 机 操 作 说 明 $”
note1 db ” F4: 交替显示剩余票及已售票          $”
note2 db ” F5: 预 留 座 注 消 处 理           $”
note3 db ” f8: 查 车 次 基 本 情 况           $”
note4 db ” F10: 消 座 号 (一 次 一 个)        $”
note5 db ” F12: 打 发 车 条 处 理             $”
note6 db ” Ctr+F3: 修 改 口 令                $”
note7 db ” Ctr+F5: 所有车预留座注消处理        $”
note8 db ” Alt+F10: 已售末号注消处理          $”
note9 db ” F2: 全 天 一 次 发 送              $”
note10 db ” J 键: 今天, M 键: 明天, H 键: 后天$”
note11 db ”    空格键: 三天轮换                 $”
note12 db ” < 键: 上一屏,  > 键: 下一屏         $”
note13 db ” END 键: 退 出 系 统                $”
inptdat3 db ”日期:  $”
inptdate db ”  查哪天车  1:今天, 2:明天, 3:后天 $”
inpdate1 db ” 哪天车  1:今天, 2:明天, 3:后天 $”
inpdate2 db ” 哪天 2:明天, 3:后天 $”
inputbus db ”  请输入车次  $”
inputst db ”  请输入站码  $”
continu db ”< F6> : 继 续 查 同 车 次 $”
contin db ”< 回车 > : 继 续 查 下 车 次 $”
ppfrec dw 0
sfinish db ”改车次已查完$”
sfinishd db ”车次已查完$”
```

Relations?

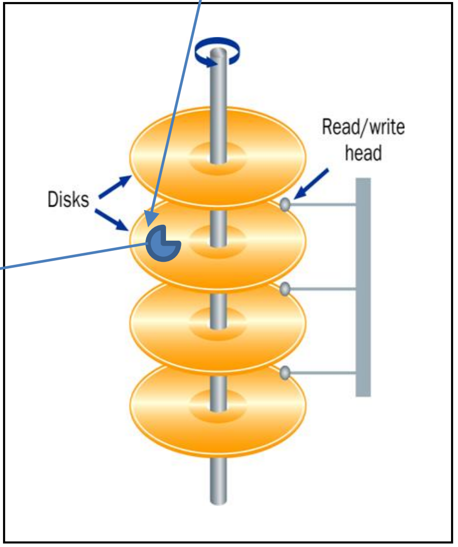
- Type and modify a program in HL or AL (by an editor)
- Translate it into object code in ML (by a compiler or assembler)
- Load it into memory for execution (by a linker and/or loader)
- Execute it on CPU (by an OS)
- Detect errors in the code (by a debugger)
- All the processes and programs are executed under and controlled by an OS such as Windows or Linux.
- IDE (Integrated Development Environment)

Your program stored on disk is loaded into memory for execution by OS (or loader particularly)

Memory address	Memory (RAM)	
0x0		
0x1	8	**DATA**
0x2	8	
0x3		
0x4	LOD R[0], 0, 0x1	**TEXT (code)**
0x5	LOD R[1], 0, 0x2	
0x6	EQ R[0], R[1], 0	
0x7	JPC 0, 0, 0x9	
0x8	**ADD R[0], R[0], R[1]**	
0x9		

Loaded into memory

Program stored on disk



Each instruction is fetched one by one into CPU for execution by decoder/control unit/interpretation

VM CPU

1. - MAR ← PC
2. - PC ← PC + 1;
3. - MDR ← MEM[MAR]
4. - IR ← MDR

Description:

Initial condition: PC = 100

Step 1: The content of PC is copied into MAR.

Step 2: PC is incremented by one the new value (101) overwrites 100.

Step 3: Instruction at location 100 is retrieved and copied into MDR.

Step 4: Instruction residing in MDR is copied into IR.

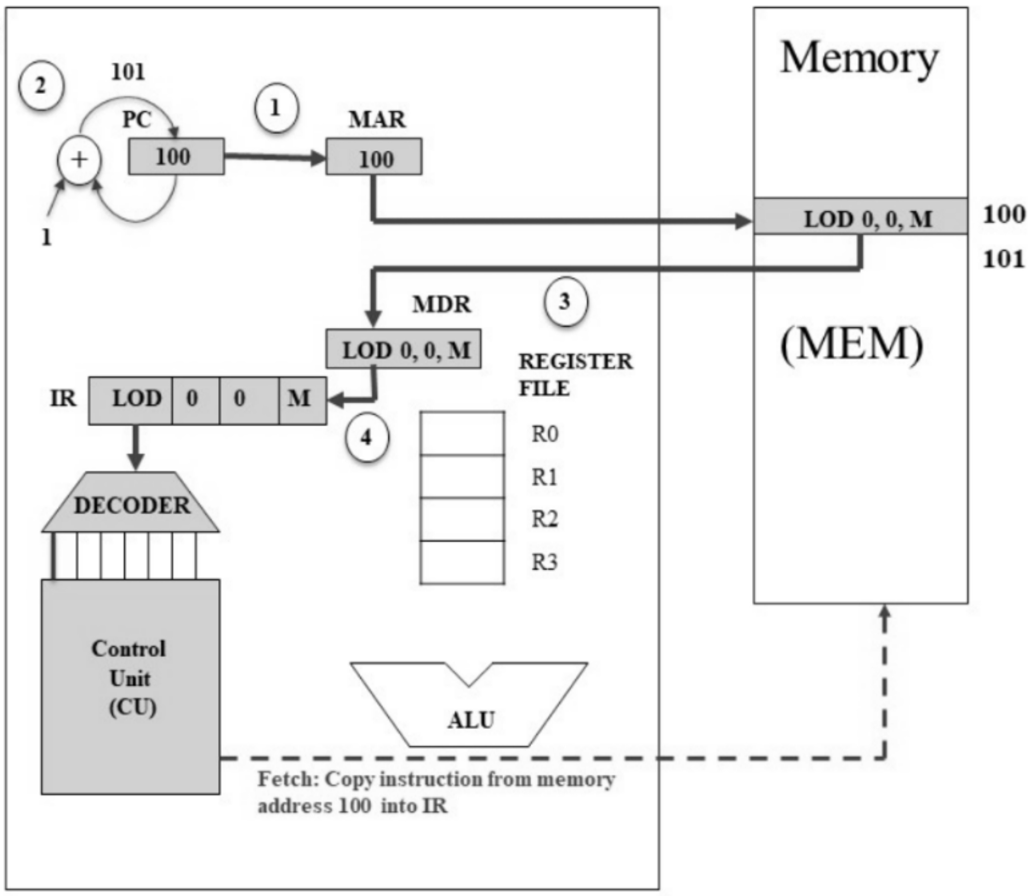


Figure 2.6. VM/0 Fetch cycle

There are many registers within CPU: PC, linkage register, index register, state word working registers

Load data from memory into CPU

1. - MAR ← IR.M
2. - MDR ← MEM[MAR]
3. - R[0] ← MDR
4. - Back to FETCH

Note that the M = 200.

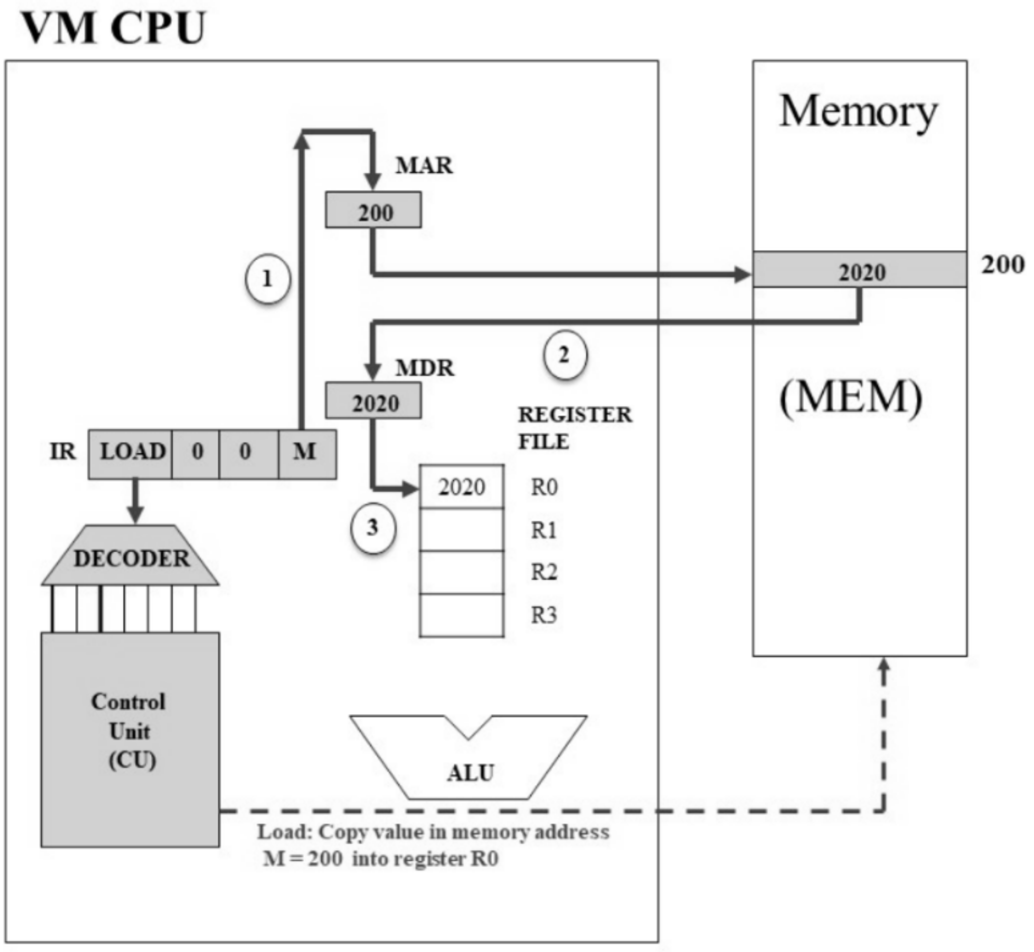


Figure 2.4. VM/0 Executing LOAD instruction

Store data (of a register) back into memory

1.- MAR ← IR.M
2.- MDR ← R[0]
3.- MDR ← MEM[MAR]
4.- Back to FETCH

Note that the M = 200.

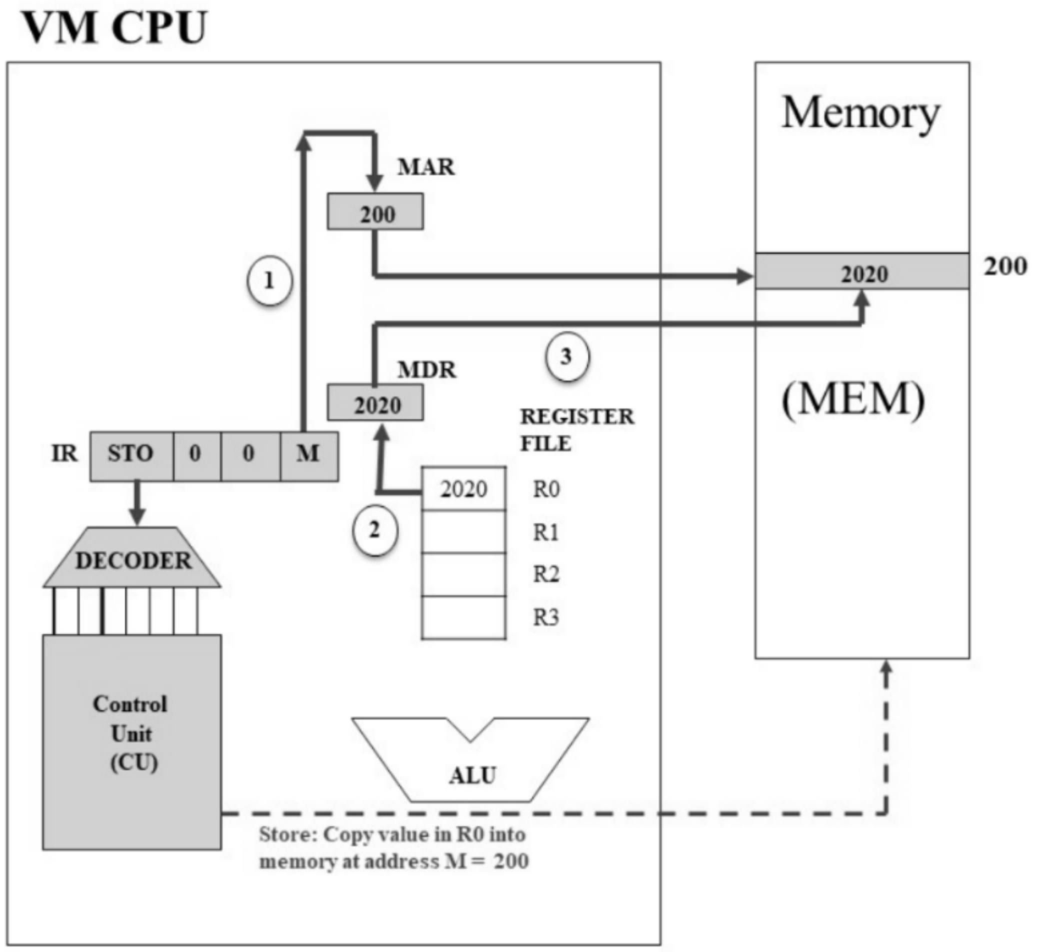


Figure 2.5. VM/0 MEM Executing STO instruction

Topics

- Assembly language and its programming
- Structure (architecture) of machines.
- Assembler
- Linker, Loader, Macro Processor
- Compiler
- Others (OS, Editor, DB, network, SE, if time allowed)

System software and Machine Architecture

- One main characteristic in which system software differs from application software: machine dependence.
  - Assembler: instruction format, addressing modes
  - Compiler: registers (number, type), machine instructions
  - OS: all of the resources of a computing system.
- Of course, some aspects of system software are machine-independent.
  - General design and logic of a assembler
  - Code optimization in a compiler
  - Linking of independently assembled subprograms.

Dilemma? And SIC

- Since for system software which is machine dependent, there is a need for real machine
- However, most real machines have certain characteristics which are unusual or unique.
- Very difficult to distinguish the features which are fundamental and those which are idiosyncrasies of a particular machines.
- In addition, so many machines
- So SIC: Simplified Instructional Computer

Sections of major chapters

- Fundamental functions
- Features depending on machine
- Features which are common and machine-independent
- Major design principles and options
  - Such as single pass or multiple passes.
- Examples of actual implementations on real machines
  - SIC, SIC/XE (Extra equipment, upward compatible)
  - CISC (Complex Instruction Set Computers) : VAX, Pentium Pro
  - RISC (Reduced Instruction Set Computers): UltraSPARC, PowerPC, Cray T3E

SIC architecture

- Memory:
  - 8-bit bytes, total 32,768 (2^{15}) bytes.
  - Word: any 3 consecutive bytes
  - Addressing:
    - any byte
    - Word: the lowest byte
- Registers:
  - Five, A, X, L, PC, SW, (0,1,2,8,9) , 24 bits each
  - A: accumulator for arithmetic operations, X: index, for address, L: Linkage, store the return address when JSUB jumps to a subroutine
  - PC:: program counter, the address of next instruction for fetch and execution, SW: Status Word, including condition code (CC)
- Data Formats:
  - Integers: 24 bits, 2's complement for negative numbers
  - Chars: 8-bit ASCII codes.
  - No floating-point numbers.
- Instruction Formats
  - Opcode (8) + x(1) + address (15), x: addressing mode
- Addressing modes:
  - Direct addressing, x=0, Target Address (TA) =address
  - Indexed addressing: x=1, TA=address+(X), address in the instruction plus Index Register
- Instruction Set:
  - Load and Store Registers: LDA, LDX, STA, STX, etc.
  - Integer Arithmetic: ADD, SUB, MUL, DIV, all these operation involve register A and a word in memory, and the result is in A.
  - COMP: compare A with a word, and set the result (<, =, >) in Condition Code (CC) in SW.
  - Conditional jump instructions: JLT, JEQ, JGT, test CC and jump accordingly
  - Subroutine linkage instruction: JSUB and RSUB. JSUB jumps to the subroutine and places the return address in L, RSUB returns by jumping to address in L.
- Input/Output:
  - IO devices: each is assigned a unique 8-bit code.
  - All input/outputs are between the rightmost 8 bits of A and a device.
  - Three instructions (with device code as operand, i.e., in the address of the instruction)
    - TD (Test Device) : whether the device is ready, condition code is set: 1: ready, 0: not ready
    - RD (Read Data) and WD (Write Data): a program must wait until a device is ready to transfer data

SIC/XE architecture

- Memory:
  - Total 1M (2^{20}) bytes. So need change the instruction formats and addressing modes.
- Registers:
  - Four more: B, S, T, F (3,4,5,6).
  - B: Base register, for addressing, S: general working register, T: general working register, F: floating-point accumulator (48 bits)
- Data Formats:
  - In addition, 48-bit floating-point numbers: s(1)+exponent(11)+fraction(36), the absolute value will be $f*2^{(e-1024)}$
- Instruction Formats
  - Mode 1 (1 byte): op (8), Mode 2 (2 bytes): op(8)+r1(4)+r2(4)
  - Mode 3(3 bytes): op(6)+n(1)i(1)x(1)b(1)p(1)e(1)+disp(12)
  - Mode 4(4 bytes): op(6)+n(1)i(1)x(1)b(1)p(1)e(1)+address(20), e=0: mode 3 and e=1: mode 4.
- Addressing modes (two new relative addressing):
  - Base relative: b=1,p=0, TA=(B)+disp ($0 \le disp \le 4095$)
  - Program –counter relative: b=0, p=1, TA=(PC)+disp ($-2048 \le disp \le 2047$)
  - b=0, p=0: TA=disp. Called direct addressing, wrt. Relative addressing.
  - In Mode 4, usually b=0 and p=0, and TA =address. Also called direct addressing., wrt. Relative addressing.
  - x=1, then (X) is added. Called indexed addressing.
  - i=1, and n=0: TA is an operand, no memory reference. So called immediate addressing.
  - i=0 and n=1: the operand is the value in the address which is the value in the address indicated by TA., called indirect addressing.
  - i=0 and n=0 or i=1 and n=1: the TA is the address of the operand. Called simple addressing.
  - Distinguish different addressing modes:
  - Indexed addressing cannot be used with immediate or indirect addressing modes.
  - i=1 and n=1: for SIC/XE with non immediate or indirect addressing,
  - i=0 and n=0: for SIC, which will keep upward compatibility. In this case, b,p,e bits will be treated as part of 15 bit address (rather than as addressing flags).
- Instruction Set (additional):
  - Load and Store to new registers, e.g., LDB, STB, etc.
  - Floating-point arithmetic: ADDF, SUBF, MULF, DIVF
  - Operations among registers: ADDR, SUBR, MULR, DIVR
  - SVC (Supervisor call): generates an interrupt that can be used for communication with OS.
- Input/Output:
  - SIO, TIO, HIO: start, test, and halt IO. These allows the parallel of IO and computing.

Machine language, Assembly Language and AL programming

- Example of SIC/XE instructions and addressing modes
  - Machine language, binary codes
  - Different addressing modes
  - Most basic operations and direct memory locations
  - Machine structure
  - Code and data both are binary.
  - (note: the five instruction 003600 is SIC one)
- Simple data movement operations for (a) SIC and (b) SIC/XE (Fig1.2-Page13)
  - Symbol (mnemonic) instruction
  - Data (its location) is referred by label.
  - Direct mapping between mnemonic instruction and machine code.
  - Later, Macro functions and subroutines.
  - Codes areas and data areas.

SIC Assembly Language Programs

- assembler directives
  - START, END, BYTE, WORD, RESB, RESW, BASE, …
- Simple arithmetic operations for (a) SIC and (b) SIC/XE
  - What is the function of the program?
- Simple looping and indexing operations for (a) SIC and (b) SIC/XE
- Another Simple looping and indexing operations for (a) SIC and (b) SIC/XE
- Simple Input and Output operations for SIC
- Simple subroutine call and record input operations for (a) SIC and (b) SIC/XE

Traditional (CISC) machines

- Large and complicate instruction set, many different instruction formats and lengths, many addressing modes.
- VAX (DEC):
  - Word, longword, quadword, octword,
  - Word alignment
  - Virtual address space—system space and process space
  - Stacks
  - Many registers, data formats, variable-length instruction formats, many addressing modes
  - I/O space is part of physical address space

CISC—Pentium Pro architecture (Intel)

- Intel x86 family
- Segments: so address is: segment:offset
- Stacks: SP:offset
- Pages
- Eight general purpose registers, several special-purpose registers, and FPU (floating-point unit) with eight 80-bit registers
- Little-endian: least significant part of a value is stored at the lowest numbered address.
- Single, double, and extended precision.
- Eight addressing modes:
  - TA=(base register) + (index register)*scale + displacement
- Over 400 instructions, including memory-to-memory
- EAX register ←→I/O port (byte, word, double word, repetition for string)

RISC

- Number of machine instructions, instruction formats, and addressing modes are small (standardized, fixed, single-cycle execution)
- Memory access are done by just load and store instructions
- All other instructions are register-to-register
- Large number of general registers.

RISC—UltraSPARC (Sun Microsystems)

- Virtual address space, pages
- Over 100 general purpose registers
- Any procedure can access only 32 registers.
- Support both big-endian and little-endian
- Three basic instruction formats with 32 bits long.
- Immediate mode, register mode, PC-relative, Register indirect with displacement, Register indirect indexed
- Less than 100 instructions
- Pipelined execution of instructions
- “atomic” instructions that can execute without allowing other memory accesses to intervene.
- No special I/O instructions, I/O port and registers are port of memory.

RISC—PowerPC (IBM)

- Virtual address space 2^{64}. Segments of fix length 256M each, pages of fix length 4096 bytes.
- 32 general purpose registers.
- Seven instruction formats with 32 bit length
- Two modes: immediate and register.
- All memory access is via load, store operation and branch instructions with three addressing modes.
- 200 instructions, some are more complicated than typical RISC such as “multiply and add” with 3 operands.
- Two I/O access modes:
  - Direct-store access: I/O port and registers as part of physical address
  - Virtual memory access: performed by normal virtual memory management hardware and software.

RISC—Cray T3E (Cray Research Inc.)

- Parallel processing
- A three dimension network of processing elements (PE)
- Each PE consists of a DEC Alpha EV5 RISC microprocessor, local memory and performance-accelerating control logic.
- Local memories form a large distributed share memory.
- Registers, Data Formats, Instruction formats, addressing modes, instruction set are as normal as RISC
- I/O: via multiple ports organized into channels. Each channel is accessible by all PEs.