DIGITAL DESIGN AND COMPUTER ORGANIZATION

Memory Operations
Instructions And Instruction Sequencing

T2: Chapter 2: 2.3-2.4

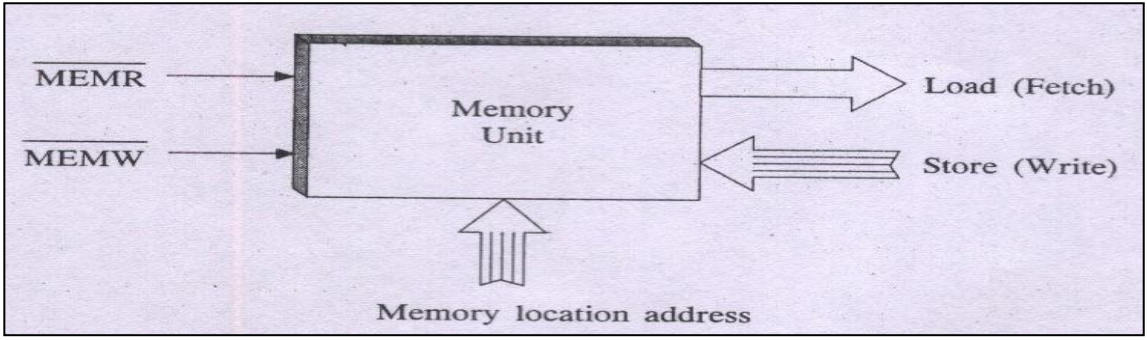
Department of Computer Science and Engineering

Memory Operations

Introduction(T2 –section 2.3)

While executing a program, two distinct operations associated with Processor-Memory interaction are:

- Load operation / Memory Read - (Instruction fetch / Operand fetch)
- Store operation / Memory Write - (Write computed results)



Memory Operations
Load (Read Or Fetch)

- Transfers a copy of the contents of a specific memory location to the processor
- Processor sends address of the desired location to memory and requests that its contents be read(Issues read signal)
- Memory reads the data and sends it to the processor
- Memory contents remains unchanged

Memory Operations
Store Or Write

- Transfers an item of info from the CPU to a specific memory location, destroying the former contents of that location
- Processor sends the address of the desired location to the memory, together with the data to be written
- Issues write signal
- Sends the data via **data bus** and write into the selected particular memory location

Instructions And Instruction Sequencing

Introduction(T2 – section 2.4)

- An instruction is a command to the processor to perform a given task on data operands.
- A program is a set of instructions that specify operations, operands & the sequence by which processing has to occur.
- Thus operation of a computer is controlled by stored program
- In general a computer must support 4 categories of operations:

1. Data Movement : To conduct I/O transfers
2. Data Storage : Across Memory and processor registers.
3. Data Processing : Arithmetic and logical operations
4. Program sequencing and control : Test and Branch.

Register Transfer Notation

- Data operands & Instructions are situated in main memory locations & processor registers.
- Like mnemonics for Opcodes, Symbolic names or label identifiers are used to name or identify such locations.
- For instances memory location names include M, LOC, A, B, C, SUM, N1, VAR1, VAR2, NUM1, etc.,
- Similarly processor register names include R0, R1, R2, R3, R4, R5, etc., and registers of I/O subsystem may be designated as DATAIN, DATAOUT etc.

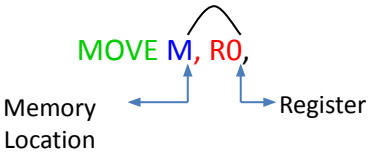
Register Transfer Notation

 While depicting operations, the contents of a memory location or a register are denoted by placing the corresponding name in a pair of square brackets.

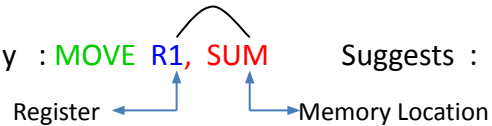
For instance : R0 □ [M]

Suggests to copy contents of memory location M to register R0

 The corresponding instruction is MOVE M, R0,



 Similarly : MOVE R1, SUM Suggests : SUM □ [R1]



 The instruction : MOVE B, LOC Means : LOC □ [B]

Register Transfer Notation

- Similar operations that involve Registers only are:

  R1 ← [R2] MOVE R2, R1

- R1 ← [R1] + [R3] ADD R3, R1

  Suggest to add contents of register R1 and register R3 and rewrite the R1 contents to store the sum.

- All such notations with leftward arrow (←) assignment operations involving register locations give rise to what is known as **Register Transfer Notation**

DIGITAL DESIGN AND COMPUTER ORGANIZATION

2. Assembly Language Notation T2:Ch2 2.4

Department of Computer Science and Engineering

Assembly Language Notation

- An assignment statement in a High Level Language Program:

S = P + Q

- Here **P**, **Q** & **S** are variables.
- **Variable name refers to *symbolic address of memory location* from which data operand can be read or written.**

S ← [P] + [Q]

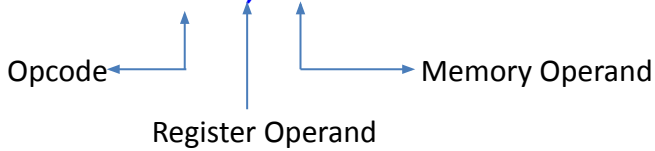
- This operation suggests contents of two memory locations **P** and **Q** are fetched from memory and transferred into processor where sum of two numbers is performed. The resulting sum is then sent back to memory and stored in memory location S.
- An instruction to this effect in Assembly Language Notation:

Add P, Q, S

Assembly Language Notation

- Consider **R1** ← **[R1] + [R3]**
- Equivalent Assembly Language instruction is **Add R3, R1**.
- Here the operation Code Mnemonic is Add
- Register Locations **R1, R3** represent *operand fields*

- Similarly, in the instruction **ADD R1, SUM**



- i.e. SUM ← [SUM] + [R1]

DIGITAL DESIGN AND COMPUTER ORGANIZATION

3. Basic Instruction Types T2:Ch2 2.4

Department of Computer Science and Engineering

Basic Instruction Types

- Three – Address Instruction
- Two – Address Instruction
- One – Address Instruction
- Zero – Address Instruction

Three – Address Instruction

Suppose we would like to use a single machine instruction to perform the following addition operation:

S □ [P] + [Q]

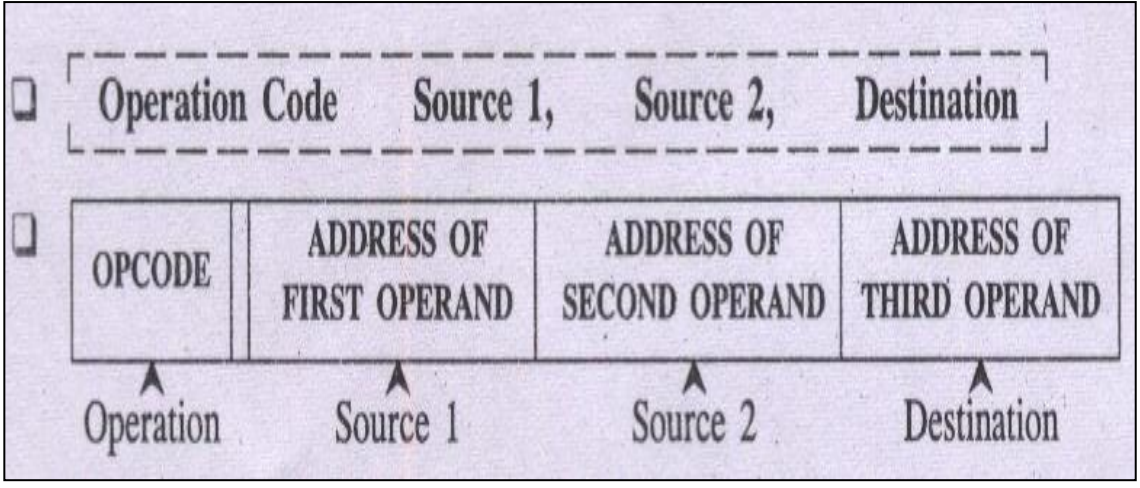
We will have to use a **three – address instruction** to carry out addition and to store the sum in a third variable S.

Then three address instruction to perform addition is:

Add P, Q, S

Basic Instruction Types

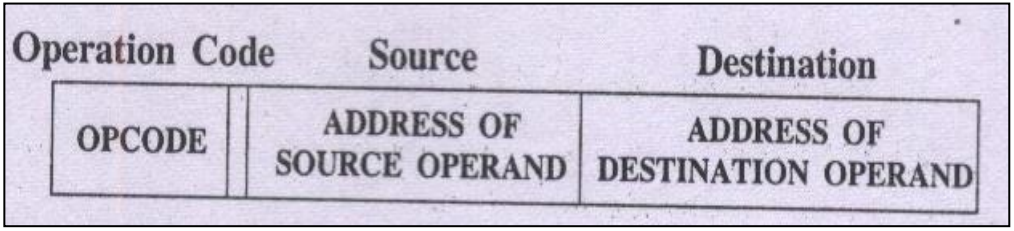
General form of three – address instruction



Basic Instruction Types

Two – Address Instruction

- Two – address instruction general format:



- Example of two – address instruction Add P, Q

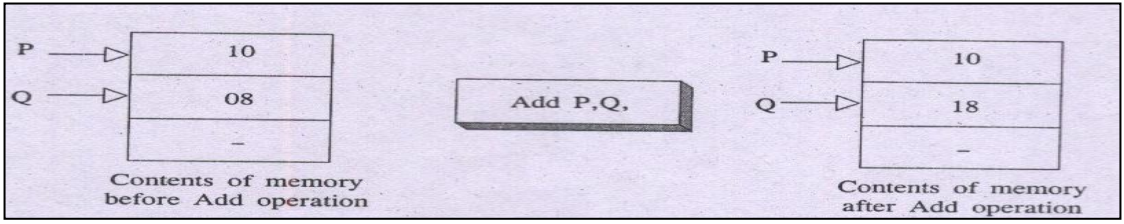
- To perform operation of addition as : Q ← [P] + [Q]

Destination Operand

Source Operand/s

Basic Instruction Types

 Here one of the operand i.e. destination operand Q acts as source as well as destination.



 To perform S □ [P] + [Q]

 Use two – address instructions sequence: MOVE Q, S
ADD P, S

Basic Instruction Types

One-Address Instruction

- One –address instruction format specify a single operand along with operation code.
- The requirement of second operand is fulfilled by an implicit processor register known as Accumulator (AC).
- Example of one-address instruction: ADD Q
- It Specify: AC ← [AC] + [Q]
- Load P Suggest AC ←[P] (Fetch P into AC)
- Store S Indicate S ← [AC] (Write AC into S)
- To perform S ← [P] + [Q]

Basic Instruction Types

One-Address Instruction

- One –address instruction format specify a single operand along with operation code.
- The requirement of second operand is fulfilled by an implicit processor register known as Accumulator (AC).
- Example of one-address instruction: ADD Q
- It Specify: AC □ [AC] + [Q]
- Load P Suggest AC □[P] (Fetch P into AC)
- Store S Indicate S □ [AC] (Write AC into S)
- To perform S □ [P] + [Q]
- Use one – address instructions : Load P; AC □[P]

  Add Q; AC □[AC] + [Q]

  Store S; S □[AC]

Basic Instruction Types

Zero – Address Instruction

- Stack – organized computer make use of a **special memory structure called push down stack to store operands**. In such computer machines it is possible to use instructions that contain only operation codes and no explicit operands.
- The name Zero – address specifies the absence of an address field of operands in machine instructions.
- Example of Zero – address instruction: ADD
- To perform operation of addition as TOS ← (P + Q)

Basic Instruction Types

Mov A,Ri is equivalent
Load A,Ri

and

Move Ri,A is equivalent to
Store Ri,A

Basic Instruction Types

If processor supports ALU operations where one data may be in memory and other in register then the instruction sequence is for

ADD A,B,C :

In processors where arithmetic operations are allowed only on operands that are in processor registers, the C = A + B task can be performed by the instruction sequence

```
Move   A,Ri
Move   B,Rj
Add    Ri,Rj
Move   Rj,C
```

In processors where one operand may be in the memory but the other must be in a register, an instruction sequence for the required task would be

```
Move   A,Ri
Add    B,Ri
Move   Ri,C
```

DIGITAL DESIGN AND COMPUTER ORGANIZATION

4. Instruction Execution And Straight-Line Sequencing

T2:Ch2 2.4

Department of Computer Science and Engineering

Instruction Execution And Straight-Line Sequencing

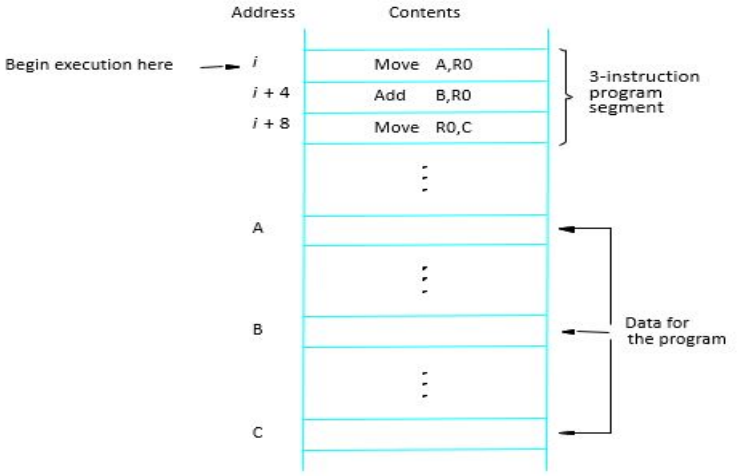


Figure 2.8. A program for C ← [A] + [B].

Assumptions:

- One memory operand per instruction
- 32-bit word length
- Memory is byte addressable
- Full memory address can be directly specified in a single-word instruction

Two-phase procedure

-Instruction fetch

-Instruction execute

Instruction Execution And Straight-Line Sequencing

- PC – Program counter: hold the address of the next instruction to be executed
- Straight line sequencing: If fetching and executing of instructions is carried out one by one from successive addresses of memory, it is called straight line sequencing.
- Major two phase of instruction execution

  - ❖ Instruction fetch phase: Instruction is fetched from memory and is placed in instruction register IR
  - ❖ Instruction execute phase: Contents of IR is decoded and processor carries out the operation either by reading data from memory or registers.

Instruction Execution And Straight-Line Sequencing

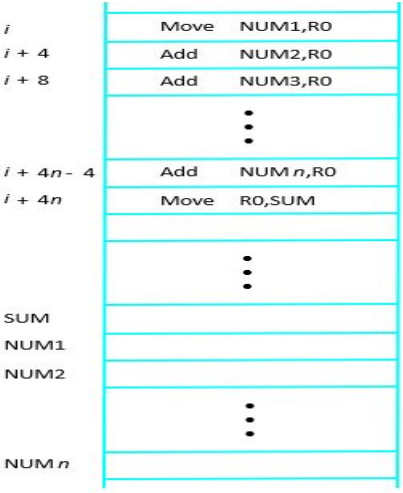


Figure 2.9. A straight-line program for adding n numbers.

DIGITAL DESIGN AND COMPUTER ORGANIZATION

5. Branching T2:Ch2 2.4

Department of Computer Science and Engineering

Branching

- Branch instruction are those which changes the normal sequence of execution.
- Sequence can be changed either conditionally or unconditionally.
- Accordingly we have conditional branch instructions and unconditional branch instruction.
- conditional branch instruction changes the sequence only when certain conditions are met.
- Unconditional branch instruction changes the sequence of execution irrespective of condition of the results.

Branching

- Consider branch instruction in a program:

Branch > 0 Loop

- Program execution continues in a straight line sequence until encountering of **Branch > 0 Loop** instruction
- Execution results in a jump to location of Loop.
- That is a branch instruction loads a new value in PC - the memory location address (loop) at which program control is to be transferred to start/resume execution.
- As a result processor is prohibited from fetching and executing next instruction in straight – line sequence.

Branching

Branch target

Conditional branch

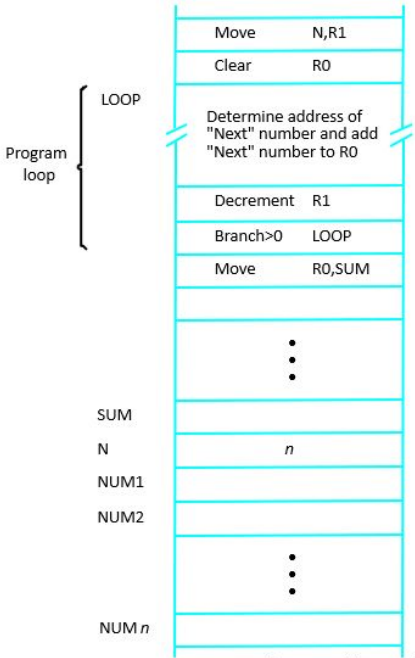


Figure 2.10. Using a loop to add *n* numbers.

Branching

Check whether number Positive or negative

```
Move a,R0

Branch >0 Pos

Move R0,R2

Pos:
Move R0,R1
```

DIGITAL DESIGN AND COMPUTER ORGANIZATION

6. Condition Codes T2:Ch2 2.4

Department of Computer Science and Engineering

Condition Codes

- The **data conditions or status, after an arithmetic or logical operation are indicated by setting or clearing the flip – flops called *flags* or *condition codes*.**
- Each flip-flop holding a data condition code is a one –bit storage cell (logic circuit) that can be set to a 1 or reset to 0 value.
- These condition codes are set/reset as a result of arithmetic and logical operations in the ALU.
- Condition code bits or flag bits are accommodated in a groups of 4 – bit, 8 bit or 16 – bit flag register or a status register in CPU.

Condition Codes

- Results of various instructions are stored for subsequent use by conditional instructions
- This is done by recording the required info in individual bits, called as condition code flags – grouped together in special processor register called as **condition code register or status register**

Condition Codes

- CONDITIONAL CODE FLAGS:

N – Negative	1 if results are Negative 0 if results are Positive
Z – Zero	1 if results are Zero 0 if results are Non zero
V – Overflow	1 if arithmetic overflow occurs 0 no overflow occurs
C – Carry	1 if carry from MSB bit 0 if there is no carry from MSB bit

Branch>0. Brach is taken when neither N or Z=1

Condition Codes

- Pentium processor makes use of following condition codes:

C (carry flag)
P (parity flag)
A (Auxiliary carry flag)
Z (zero flag)
S (sign flag)
O (over flow flag)

Condition Codes

List the steps needed to execute the machine instruction Add LOC, R0 in terms of transfers between memory and processor and some simple control commands. Assume that the instruction itself is stored in the memory at location INSTR and that this address is initially in register PC.

Condition Codes

- Transfer the contents of register PC to register MAR
- Issue a Read command to memory, and then wait until it has transferred the requested word into register MDR
- Transfer the instruction from MDR into IR and decode it
- Transfer the address LOCA from IR to MAR
- Issue a Read command and wait until MDR is loaded
- Transfer contents of MDR to the ALU
- Transfer contents of R0 to the ALU
- Perform addition of the two operands in the ALU and transfer result into R0
- Transfer contents of PC to ALU
- Add 1 to operand in ALU and transfer incremented address to PC

Condition Codes

Repeat the problem for machine instruction
Add R1,R2,R3

Condition Codes

- Transfer the contents of register PC to register MAR
- Issue a Read command to memory, and then wait until it has transferred the requested word into register MDR
- Transfer the instruction from MDR into IR and decode it
- Transfer contents of R1 and R2 to the ALU
- Perform addition of two operands in the ALU and transfer answer into R3
- Transfer contents of PC to ALU
- Add 1 to operand in ALU and transfer incremented address to PC

Condition Codes

Give a short sequence of machine instructions for the task: “Add the contents of memory location A to those of location B, and place the answer in location C.”

Instructions Load LOC, R_i

and Store R_i, LOC

are the only instructions available to transfer data between the memory and general purpose register R_i. Do not destroy the contents of either location A or B.

Condition Codes

Load A,R0
Load B,R1
Add R0,R1
Store R1,C

Condition Codes

Suppose that move and add instructions are available in the format
Move/Add loc1,loc2

Loci can be either memory or reg. Is it possible to use fewer instructions?
Give the seq

Condition Codes

Move B,C
Add A,C

DIGITAL DESIGN AND COMPUTER ORGANIZATION

7. Generating Memory Addresses T2:Ch2 2.4

Department of Computer Science and Engineering

Generating Memory Addresses

- The instruction block at **LOOP** needs to add a different number from a list during each iteration.
- The **Add** instruction must refer to a different memory address during each pass through the loop.
- The memory operand address cannot be directly given in a single **Add** instruction because it would require modification on each pass.
- A possible solution is to use a processor register, such as **Ri**, to hold the memory address.
- **Ri** can be initialized with the address **NUMI** before the loop and incremented by 4 after each iteration to point to the next memory address.
- This situation highlights the need for flexible ways to specify operand addresses.
- These methods are known as **addressing modes**, which vary in implementation but share common underlying concepts across different computers.

Think about it

- Write assembly code in three two and one address format for SUB A,B,C
- Write assembly code in three two and one address format for A=B+C-D

MCQ

A processor executes instructions in **straight-line sequencing** unless:
a) A Load instruction is encountered
b) A Store instruction is encountered
c) A Branch instruction is encountered
d) A Condition Code is set

Which condition flag is set to 1 if an arithmetic operation results in a value too large for the number of bits available?
a) Zero (Z)
b) Carry (C)
c) Overflow (V)
d) Sign (N)

MCQ

A processor executes instructions in **straight-line sequencing** unless:
a) A Load instruction is encountered
b) A Store instruction is encountered
c) A Branch instruction is encountered
d) A Condition Code is set
Answer: c) A Branch instruction is encountered
Explanation: Branch instructions alter the normal program counter (PC) sequence

Which condition flag is set to 1 if an arithmetic operation results in a value too large for the number of bits available?
a) Zero (Z)
b) Carry (C)
c) Overflow (V)
d) Sign (N)
Answer: c) Overflow (V)
Explanation: Overflow flag is set when the signed result exceeds representable range

MCQ

- Which one of the following instruction formats uses the **accumulator (AC)** implicitly as one operand?

a) Zero-address instruction b) One-address instruction c) Two-address instruction d) Three-address instruction

- Consider the sequence of instructions:

Load P
Add Q
Store S
This corresponds to which instruction format?
a) Three-address
b) Two-address
c) One-address
d) Zero-address

MCQ

- Which one of the following instruction formats uses the **accumulator (AC)** implicitly as one operand?

a) Zero-address instruction b) One-address instruction c) Two-address instruction d) Three-address instruction

Answer: b) One-address instruction
Explanation: In one-address instructions, one operand is explicitly given, while the accumulator (AC) is the implicit second operand

- Consider the sequence of instructions:

Load P
Add Q
Store S
This corresponds to which instruction format?
a) Three-address
b) Two-address
c) One-address
d) Zero-address

Answer: b) One-address instruction

THANK YOU

Team DDCO
Department of Computer Science