Control Design

0.1 Introduction

Each machine instruction is executed as a sequence of lower level operations, for example, for a commonly used memory reference instruction the lower level operations are:

1. dispatch of address to address bus,
2. fetch the instruction,
3. decode it,
4. fetch operands,
5. execute it, and
6. store the result.

These operations are called *micro-operations.* The function of Control unit is to initiate sequences of micro-operations. The complexity of the digital system is derived from the variety of number of sequences of micro-operations that are performed.

While performing these micro-operations, the control can be viewed as *state machine* that changes from one state to another state in every clock cycle, depending on contents of various registers. The output of machine, i.e. control unit (CU) are control signals: $c_0 \dots c_n$, which cause the micro-operations to occur in a definite sequence.

The sequence of operations carried out is decided by *wiring of logic* elements, hence the name *hard-wired control.* The controller that uses this approach can operate at high speed, but the cost and complexity of implementation are limitations.

An alternative approach is micro-programmed control, where control signals are generated by a program similar to machine language.

0.2 Micro-operations

An instruction cycle can be viewed as:

Instruction cycle = *Fetch Cycle* + *Execute cycle.*

Each of the fetch and execute cycles are performed as number of micro-operations. Let the micro-operations be represented by clock cycles: $t_1, t_2, \ldots, t_n$. A *fetch* cycle may consist:

$t_1 : MAR \leftarrow (PC)$

$t_2 : MBR \leftarrow Memory$

$PC \leftarrow (PC) + \ell$; ($\ell$ = length of this instruction)

$t_3 : IR \leftarrow (MBR(opcode))$

This is followed with execute cycle, which varies from one instruction to other instruction. If it is *indirect address* instruction, the execution follows after fetching of the address.

Following are the operations for fetching effective address after having fetched the Instruction along with the indirect address, called *indirect cycle.*

$t_4 : MAR \leftarrow (MBR(indirectAddr))$

$t_5 : MBR \leftarrow Memory$

$t_6 : MAR \leftarrow (MBR(Addr))$

$t_7 : MBR \leftarrow Memory$

Following are the micro-operations for interrupt-cycle and execute-cycles.

Interrupt Cycle:

$t_1 : MBR \leftarrow (PC)$; save current PC value in MBR

$t_2 : MAR \leftarrow Save\ addr$; for saving PC

$PC \leftarrow Subroutine\ addr$; call subroutine

$t_3 : memory \leftarrow (MBR)$; save PC in stack

Following are micro-operations for the *Execute cycle* for performing an addition. These micro-operations are less predictable.

Copyrighted Material.

Instruction: $ADD\ R_1, X; R_1 \leftarrow R_1 + X$

$t_1 : MAR \leftarrow (IR(Addr))$

$t_2 : MBR \leftarrow Memory$

$t_3 : R_1 \leftarrow (R_1) + (MBR)$; The contents of R_1 and the one pointed by x are added)

ISZ X: Increment and skip next instruction, if result zero (PDP-8 memory reference instruction). It is used for loop control with initial value of loop counter as negative. The following section of the code shows the use of ISZ instruction.

```
initialize counter x to -15 (say)
...
loop: loop-body
      isz x
      jmp loop
      instruction
      ...
      hlt
```

The sequence of micro-operations for $ISZ\ x$ are:

$t_1 : MAR \leftarrow (IR(Addr))$; address of x goes in MAR

$t_2 : MBR \leftarrow Memory$; fetch x in MBR

$t_3 : MBR \leftarrow (MBR) + 1$; increment x

$t_4 : Memory \leftarrow (MBR)$; save x

if ((MBR)=0) then (PC = (PC)+1); performed as single micro-operation.

The representation (PC) indicates the contents of PC.

Subroutine call: The subroutine call for PDP-8 machine is implemented by *BSA X*, i.e. *Branch-and-save-address Instruction*. It returns the address of next instruction is saved at X, and execution start at $X + 1$. Figure 1 shows the subroutine call using BSA instruction.

Following are the sequence of micro-operations for BSA.

$t_1 : MAR \leftarrow (IR(addr))$; move X into MAR

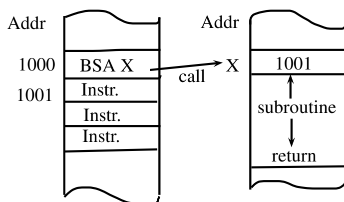

Figure 1: Subroutine call using BSA Instruction.

$MBR \leftarrow (PC)$; move 1001 into MBR

$t_2 : PC \leftarrow (IR(addr))$; move X into PC

$Memory \leftarrow (MBR)$; save MBR (i.e 1001) at X

$t_3 : PC \leftarrow (PC) + 1$; point PC to called subroutine's code

Subsequent to these micro-operations, the subroutine instructions are executed. On the completion of the subroutine, the return instruction loads the value at x into the PC, which returns the control back to original code.

0.3 Instruction cycle

Each phase of instruction is decomposed into sequence of elementary operations (micro-ops). By defining the operations of processor in terms of these micro-ops, we can define what control unit should exactly do. Implementation of micro-ops is nothing but the design of Control unit. All micro-operations fall into one of these categories:

- *R-R:* register-register data transfer
- *R-bus:* register-internal data bus data-transfer
- *R-extbus:* register-external bus data-transfer
- Performs arithmetic and logic operations using registers
- Following are the two functions of CU:
    1. *Sequencing:* Causes the processor to step through sequence of micro-ops,
    2. *Execution:* CU causes each micro-operation to be performed.

 Copyrighted Material.

0.3.1 Control Signals

The CU must comprise the logic to perform the *sequencing* and *execution* of micro-operations. A typical case of simplified control unit is shown in the figure 2.

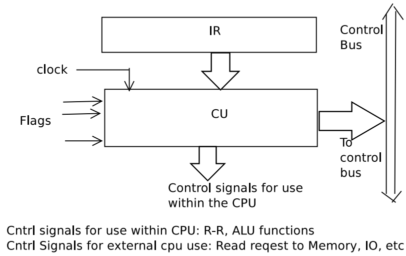

Figure 2: Control unit

The *Flags* are required by CU to determine the status of processor, and outcome of previous operations. For example, in the ISZ instruction, the CU will increment PC if zero *flag* is set. The output of CU are classified into two categories:

- Control signals that are used within CPU,
- Control signals that are used to control the bus.

All these control-signals are directly applied to the binary logic gates.

Following are the examples of control signals generated at times t_1, t_2, etc., which corresponds to the order of CPU clock pulses. Some of the control signals are also indicated in figure 3, as indicated by lines $c_0 \dots c_{12}$.

A fetch cycle may comprise following micro-operations(see figure 3):

$t_1 : MAR \leftarrow (PC)$; open the gate(s) to let data to move: c_6

$t_2 : MBR \leftarrow Memory$; open the gate(s): c_2

$PC \leftarrow (PC) + 1$; not shown in this fig.

$t_3 : IR \leftarrow (MBR)$; open the gate(s): c_{12}

0.4 Hardwired Control

Each control signal is generated as function of inputs consisting of flags and output from Instruction decoder, hence, a control signal c_i can be

Copyrighted Material.

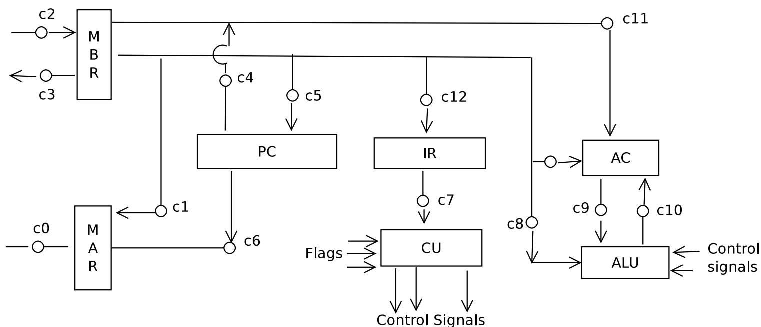

Figure 3: Control signals $c_0 : c_{12}$ control the data flow.

represented as, $c_i = f(signals, flags, set\ of\ timing\ signals\ T_i)$. These control lines are making internal control as well as the external control bus. In hard-wired control, CU acts a *state machine*. The CU makes use of decoded *opcode* from Instruction Register (IR) to generate signals for fetching, decoding, and execution of each instruction. The figure 4 demonstrates, how each control signal c_i is generated through control unit, due to the combined effect of many inputs.

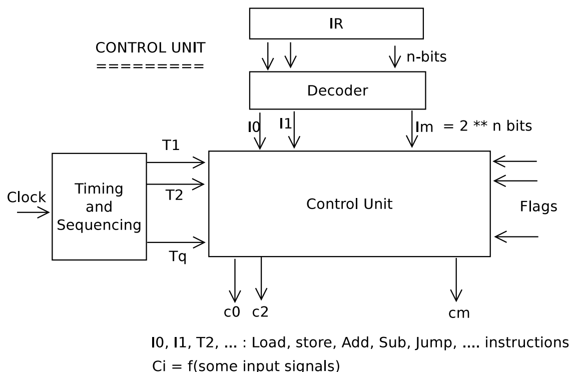

Figure 4: Generating Control signals C_i through Control Unit.

0.5 Timing and Sequencing of Micro-operations

Let the control signals be p, q. And, let the state $pq = 00$ indicates the *fetch cycle* of the instruction, $pq = 01$ is for *indirect cycle*, $pq = 10$

 Copyrighted Material.

indicates *execute cycle*, and $pq = 11$ for *interrupt cycle*. Then, a control line, say c_2 (to allow to read data from external bus) can be generated by: $c_2 = \bar{p}\bar{q}T_2 + \bar{p}q.T_2$. See figure 3 for the position of c_2. When the control lines are generated in this manner, it is called *hardwired control* (HWC).

Let $T_1, T_2, T_3, \ldots,$ are different *timing signals*, each corresponds to CPU cycle. A collection of timing-signals is called a *machine cycle*, shown as M_1, M_2, etc, in figure 5. Let there be control signals I_0, I_1 from *Instruction decoder*, representing LDA, ADD opcodes, respectively. The lines I_0, I_2 signal as memory-read when the instruction is executed. Given this, the control signal C_2 (see figure 5) can be generated by the expression,

$$c_2 = \bar{p}\bar{q}.T_2 + \bar{p}q.T_2 + p\bar{q}(LDA + ADD).T_2 \tag{1}$$

In addition to the proper control signals, the clock cycle must be long enough to allow for propagation of signals along the data bus.

The hard-wired control has the characteristics of being complex, fast, difficult to design, and difficult to modify. Also, lots of optimization is required during the implementation phase.

The figure 5 shows the timing signals, including the control signals for the 8085 microprocessors "OUT" instruction. The format of this instruction is

$$OUT\ n$$

where n is a 8-bit port address where accumulator A's contents are to be written. The Z is output temporary register of ALU. What ever is single input to accumulator, it goes to Z because the accumulator does not hold a value.

0.6 Micro-Programmed Control

In *micro-programmed control*, instead of generating the control signals from hardware logic, the control signals are generated by a program similar to machine language, which is stored in a memory inside the CPU, called *control memory*. Each word, called *control-word*, when fetched from control memory, the bits of that word generates control lines of true / false, from the bits 1/0. The word's individual bits, called *binary variables*, represent various control signals. When a binary variable is in the true state, the corresponding micro-operation is performed. In a *bus system* control signals specify micro-operation as group of bits that select the path in *multiplexers*, *decoders*, and *ALU*.

Copyrighted Material.

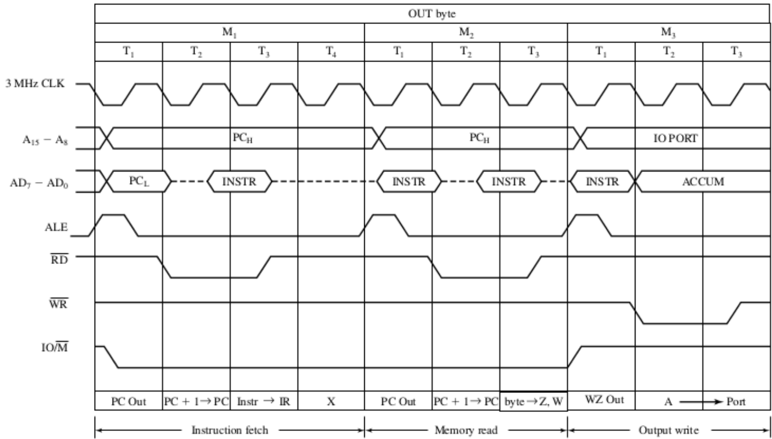

Figure 5: Timing diagram of Instruction of 8085 OUT instruction.

The number of micro-operations are finite. At one time only certain number of micro-operations are initiated. A sequence of control words corresponding to control sequence of a machine instruction constitute the *micro-routine* for that instruction, and individual control words are referred to as *micro-instruction*.

Micro-routines reside in *control memory*, and its micro-instructions are accessed by micro-program counter (μPC). Every time a new machine language instruction is loaded into IR, the μPC is initiated with new address by starting address generator. This causes the successive micro-instructions to be read from *Control Memory*.

The Control words, i.e.micro-instructions, can be programmed to perform various operations on the components of the system.Control words are interpreted by the microprogram control unit. A microinstruction a may specify one or more micro-operations. The figure 6 shows a typical micro-program control unit.

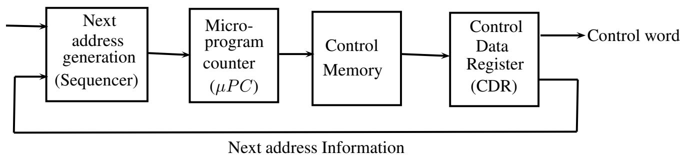

Figure 6: Microprogrammed Control organization.

Copyrighted Material.

A data register, called *Control Data Register* (CDR) loads the micro-instruction from control-memory into micro-program instruction register.

Generally, the micro-programs are *static*, i.e., they are permanently stored in the control memory. Changing a micro-program changes the instruction set for a CPU. A *Dynamic microprogram* is loaded initially at the start of system in a RAM.

0.6.1 Control Unit Operation

On execution of a micro-instruction, it determines the next address of control-word to be fetched. Since a micro-program is not hardware to generate the control signals, a microprogrammed system is far more flexible than a hard-wired control. But, with some cost reduced, the speed of a micro-programmed system is slower than a hardwired system.

The *address sequencing* is carried out as follows. Each machine instruction has its own micro-program routine in control-memory to generate micro-operations. A *mapping* process translates the machine instruction code into micro-program routine's address. Micro-instructions are sequenced by incrementing micro-program counter(μPC), or computed by status bits in the case of jump instructions in the micro-program routine. After completion of execution of a control routine, control is transferred to a fetch routine. This is by unconditional branch microinstruction to first address of the fetch routine. This fetches the next machine instruction from the main memory.

The conditional branching in a micro-program is decided by status bits(C, S, Z, etc), which are stored in some register. The status bits together with branch address, control the conditional branch decision generated by branch logic. The figure 7 shows the selection of next micro-instruction address.

The tasks performed by a Microprogrammed Control Unit are as follows:

1. Microinstruction sequencing
2. Microinstruction execution
3. Must consider both together

The above order has advantages that it simplifies the design of control unit, it is cheaper, less error-prone, but has the disadvantage of being slower.

For micro-instruction sequencing can be carried out using *two address*, *one address*, and *variable address* format. In two address format, one

Copyrighted Material.

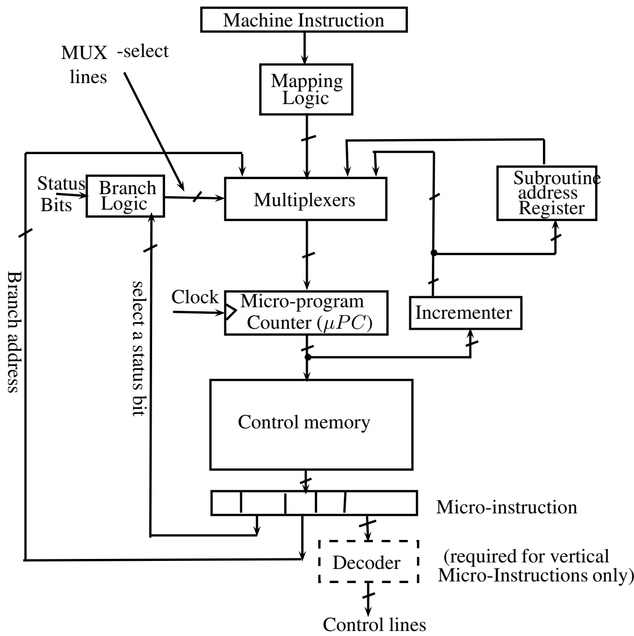

Figure 7: Selection of address for control memory.

address is default address, and other is conditional branch address. This however, wastes a lot of space.

In one address format, a branch address is provided is always provided. This may be conditional or unconditional.

In the variable address format, only branch instructions have address, and others not.

Following are the design considerations for micro-programmed control:

1. Size of microinstructions

2. Address generation time

   It is determined by instruction register(Once per cycle, after instruction is fetched)

3. Next sequential address(common in most designs)

4. Branches(both conditional and unconditional branches)

 Copyrighted Material.

Micro-Instruction encoding

There are number of encoding techniques used for encoding of micro-instructions.

- **Functional Encoding:** In this technique, each field controls some functions. For example, load accumulator, load ALU operands, compute next program counter value, etc.
- **Resource Encoding:** In this approach, each field controls some resource, like ALU, memory, etc.

In a vertical format, a 8-bit microinstruction may be split into sub-fields as *type, operations, register*, each of size 3-bit, 3-bit, 2-bit, respectively.

In a horizontal format, there may be large number of fields, like, *register transfer, memory operation,sequence operation, ALU operation, register selection*, etc. As along as resources are not common, many operations can be done parallel in the horizontal instruction format.

0.7 Types of Micro-programmed Controls

In a micro-programmed control, the Machine instructions' fetch execute cycle cause the machine instructions to be executed at CPU, while the Micro-instructions' fetch-execute cycle produce control signals for data path. A μProgram is stored in control memory, like ROM, PROM, or EPROM.

In fact, a Micro-program defines the instruction set architecture of a computer, as it defines each of the machine instruction. To change instruction set, we reload control memory by different micro-program.

Micro-programmed control is classified into two categories: (1) *Vertical micro-programming*, where each micro-instruction specifies single (or few) micro-operations to be performed, and (2) *Horizontal micro-programming*, where each micro-instruction specifies many different micro-operations to be performed in parallel.

0.7.1 Horizontal Micro-Programming

The figure 8 shows horizontal micro-program instruction. The horizontal micro-program control divides control signals into disjoint groups, and implements each group as separate field in memory word. Due to this feature, the horizontal micro-programmed control supports reasonable levels of parallelism without too much complexity.

Copyrighted Material.

The longer instructions (control words) have good potential of parallelism, and requires lesser encoding of micro-instructions. The control memory is also larger, hence cost of the system shall be higher.

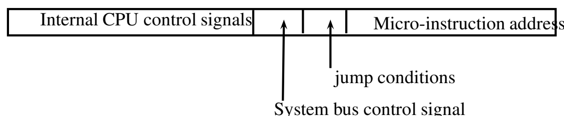

Figure 8: Horizontal Control Instruction.

0.7.2 Vertical Micro-Programming

In a vertical micro-programming, width of a micro-instruction is narrow, control signals are encoded into $\lg_2 n$ bits of the number of control lines, and there is a limited ability to express parallelism. There is a considerable encoding of control information. This requires external memory word decoder to identify the exact control line being manipulated.

The figure 9 shows the vertical micro-programmed control. Though compact, it does not support parallelism. The word length of control-word is small compared to horizontal micro-programmed control, as it supports a higher level of encoding of micro-instructions. Hence, the cost of control memory is less, with over all system as slower, with lesser cost.

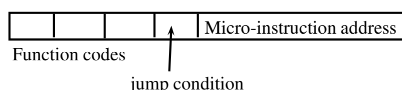

Figure 9: Vertical Microprogrammed Control.

0.7.3 Control Memory

There are following criteria for microprogram word length: Maximum number of simultaneous micro-operations should be supported, the choice of control depends on the way control information is represented or encoded, and the way in which the next micro-instruction address is specified. The figure 10 shows the control memory indicating the control-program structure.

The figure 11 shows a "Pure hardwired control v/s Pure micro-programmed control" design sequences.

 Copyrighted Material.

Initially, the hardwire control is represented using *finite state machine* with state and transitions, where as in micro-programmed control, a micro-program routine instructions realize these transitions. In micro-program control, the micro-program counter causes switching in state, due to fetching of next micro-instruction, while while in hardwired control (HWC) it is done through a state transition function. The third state is logic equation in HWC, and truth table in micro-programmed control. In the last stage, HWC control is implemented through programmable logic arrays, while micro-program is stored in ROM memory.

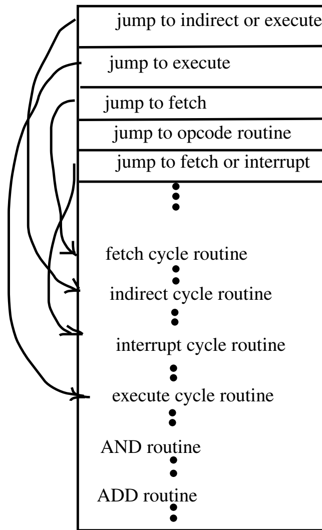

Figure 10: Control Memory.

0.8 Hardwired v/s Micro-programmed Control

The micro-programs are simple to design, as it requires to encode the control lines and addresses in the micro-instructions. Since it is a program, it is *flexible*, can adapt to changes in organization with time and technology. Also, it allows changes which are late in the design cycle, and even the changes can be done in the field. Powerful instruction set can be used, which may require larger size of control memory, as well complex instructions.

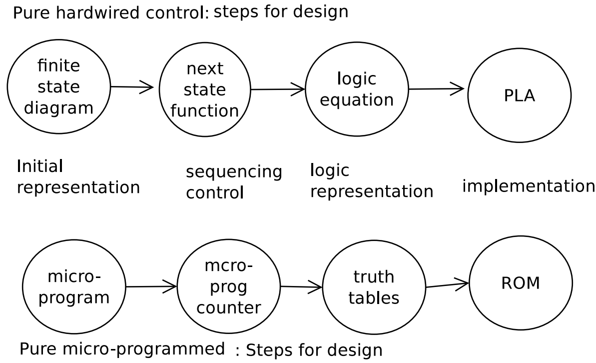

Figure 11: Hardwired v/s m-programmed control design steps.

The micro-programmed control provides the *generality* in the design, due to which even multiple instruction set can be used for the same machine. The instruction set can be tailor-made for for specific application.

It is easy to design micro-programmed control which is backward *compatible* in a family of processors. Even, in different organizations, same instruction set can be used.

A micro-programmed control may some-times be costly, as it may require special tools for micro-program development and micro-program compiler.

Exercises

1. Based on the instruction BSA (figure 1), answer the following questions:
   (a) Can the subroutines be nested using this?
   (b) Can the subroutine be called recursively using this approach?
   (c) What is disadvantage (if any)of this approach to subroutine call?

2. A microprogrammed control organization shown in figure 6 has the following propagation delay times: 40 ns to generate next address, 10 ns to transfer the address into the control address register (μPC), 40 ns to access the control memory ROM, and 40 ns to perform the required micro-operations specified by the control word. What is the maximum frequency that the control can use? What is the maximum clock frequency be if the control data register is not used?

 Copyrighted Material.

3. The system shown in figure 7 uses a control memory of 1024 words of 32 bits each. The micro-operations field has 16-bits.
   (a) How many bits are there in the branch address field and the select field?
   (b) If there are 16-status bits in the system, how many bits of the branch logic are used to select a status bit?
   (c) How many bits are left to select an input for the multiplexers?
4. The control memory shown in figure 7 has 4096 words of 24 bits each.
   (a) How many bits are there in the control address register?
   (b) How many bits are there in each of the four inputs shown going into the multiplexers?
   (c) What is the number of inputs in each multiplexer and how many multiplexers are needed?
5. Explain how the mapping of from an instruction code to a microinstruction address can be done by means of a read only memory?
6. Write a symbolic micro-program routine for the ISZ X instruction.
7. Show how a 9-bit micro-operation field in a microinstruction can be divided into subfields to specify 46 micro-operations. What is largest number of micro-operations that can be specified in one micro-instruction?
8. A computer has 8 registers, an ALU with 32 operations, and a shifter with 8-operations, all connected to a common bus system.
   (a) Formulate a control-word for a micro-operation.
   (b) Specify the number of bits in each field of control word and give a general encoding scheme.
   (c) Show the bits of control word that specify the micro-operations $R_4 \leftarrow R_5 + R_6$.
9. RISC should use what control, i.e., hardwired or microprogrammed?

Copyrighted Material.

Copyrighted Material.

Bibliography

[1] John P. Hayes, "Computer Architecture and Organization", 2nd Edition, McGraw-Hill, 1988.

[2] William Stalling, "Computer Organization and Architecture", 8th Edition, Pearson, 2010.

[3] M. Morris Mano, "Computer System Architecture", 3rd Edition, Pearson Education, 2006.

[4] Carl Hamacher, Zvono Vranesic, and Safwat Zaky, "Computer Organization", , 5th edition, McGrawhill Education, 2011. (chapter 7)

[5] http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-823-computer-system-architecture-fall-2005/lecture-notes/