# BASIC COMPUTER ORGANIZATION AND DESIGN

* **Instruction Codes**
* **Computer Registers**
* **Computer Instructions**
* **Timing and Control**
* **Instruction Cycle**
* **Memory Reference Instructions**
* **Input-Output and Interrupt**
* **Complete Computer Description**
* **Design of Basic Computer**
* **Design of Accumulator Logic**

# INTRODUCTION

* **Every different processor type has its own design (different registers, buses, microoperations, machine instructions, etc)**
* **Modern processor is a very complex device**
* **It contains**
  + **Many registers**
  + **Multiple arithmetic units, for both integer and floating point calculations**
  + **The ability to pipeline several consecutive instructions to speed execution**
  + **Etc.**
* **However, to understand how processors work, we will start with a simplified processor model**
* **This is similar to what real processors were like ~25 years ago**
* **M. Morris Mano introduces a simple processor model he calls the *Basic Computer***
* **We will use this to introduce processor organization and the relationship of the RTL model to the higher level computer processor**

# THE BASIC COMPUTER

* **The Basic Computer has two components, a processor and memory**
* **The memory has 4096 words in it**
  + **4096 = 212, so it takes 12 bits to select a word in memory**
* **Each word is 16 bits long**

**CPU RAM**

**0**

**15 0**

**4095**

# INSTRUCTIONS

* **Program**
  + **A sequence of (machine) instructions**
* **(Machine) Instruction**
  + **A group of bits that tell the computer to *perform a specific operation***

**(a sequence of micro-operation)**

* **The instructions of a program, along with any needed data are stored in memory**
* **The CPU reads the next instruction from memory**
* **It is placed in an *Instruction Register* (IR)**
* **Control circuitry in control unit then translates the instruction into the sequence of microoperations necessary to implement it**

# INSTRUCTION FORMAT

* **A computer instruction is often divided into two parts**
  + **An *opcode* (Operation Code) that specifies the operation for that instruction**
  + **An *address* that specifies the registers and/or locations in memory to use for that operation**
* **In the Basic Computer, since the memory contains 4096 (=**

212) words, we needs 12 bit to specify which memory address this instruction will use

* **In the Basic Computer, bit 15 of the instruction specifies the *addressing mode* (0: direct addressing, 1: indirect addressing)**
* **Since the memory words, and hence the instructions, are 16 bits long, that leaves 3 bits for the instruction’s opcode**

**Instruction Format**

**15 14 12 11 0**

**I Opcode**

**Address**

**Addressing mode**

# ADDRESSING MODES

* **The address field of an instruction can represent either**
  + **Direct address: the address in memory of the data to use (the address of the operand), or**
  + **Indirect address: the address in memory of the address in memory of the data to use**

**Direct addressing Indirect addressing**

**22**

**457**

**0 ADD**

**Operand**

## +

**457**

**35**

**300**

|  |  |  |
| --- | --- | --- |
| **1** | **ADD** | **300** |
|  | | |
| **1350** | | |
|  | | |
| **Operand** | | |
|  | | |

**1350**

## +

**AC AC**

* **Effective Address (EA)**

– **The address, that can be directly used without modification to access an operand for a computation-type instruction, or as the target address for a branch-type instruction**

# PROCESSOR REGISTERS

* + **A processor has many registers to hold instructions, addresses, data, etc**
  + **The processor has a register, the *Program Counter* (PC) that holds the memory address of the next instruction to get**
    - **Since the memory in the Basic Computer only has 4096 locations, the PC only needs 12 bits**
  + **In a direct or indirect addressing, the processor needs to keep track of what locations in memory it is addressing: The *Address Register* (AR) is used for this**
    - **The AR is a 12 bit register in the Basic Computer**
  + **When an operand is found, using either direct or indirect addressing, it is placed in the *Data Register* (DR). The processor then uses this value as data for its operation**
  + **The Basic Computer has a single *general purpose register* – the *Accumulator* (AC)**

# PROCESSOR REGISTERS

* + **The significance of a general purpose register is that it can be referred to in instructions**
    - **e.g. load AC with the contents of a specific memory location; store the contents of AC into a specified memory location**
  + **Often a processor will need a scratch register to store intermediate results or other temporary data; in the Basic Computer this is the *Temporary Register* (TR)**
  + **The Basic Computer uses a very simple model of input/output (I/O) operations**
    - **Input devices are considered to send 8 bits of character data to the processor**
    - **The processor can send 8 bits of character data to output devices**
  + **The *Input Register* (INPR) holds an 8 bit character gotten from an input device**
  + **The *Output Register* (OUTR) holds an 8 bit character to be send to an output device**

# BASIC COMPUTER REGISTERS

**Registers in the Basic Computer**

**11 0**

**PC**

**11 0**

**AR**

**Memory 4096 x 16**

**15 0**

**IR**

**15 0**

**TR**

**15 0**

**DR**

**CPU**

**7**

**OUTR**

**0 7 0**

**INPR**

**15 0**

**AC**

**List of BC Registers**

|  |  |  |  |
| --- | --- | --- | --- |
| **DR** | **16** | **Data Register** | **Holds memory operand** |
| **AR** | **12** | **Address Register** | **Holds address for memory** |
| **AC** | **16** | **Accumulator** | **Processor register** |
| **IR** | **16** | **Instruction Register** | **Holds instruction code** |
| **PC** | **12** | **Program Counter** | **Holds address of instruction** |
| **TR** | **16** | **Temporary Register** | **Holds temporary data** |
| **INPR** | **8** | **Input Register** | **Holds input character** |
| **OUTR** | **8** | **Output Register** | **Holds output character** |

# COMMON BUS SYSTEM

* **The registers in the Basic Computer are connected using a bus**
* **This gives a savings in circuitry over complete connections between registers**

# COMMON BUS SYSTEM

**S2**

**Memory unit 4096 x 16**

**S1 S0**

**Address**

**Bus 7**

**Write**

**Read**

**AR 1**

**ALU**

**LD INR CLR**

**PC 2**

**LD INR CLR**

**DR 3**

**LD INR CLR**

**E**

**AC 4**

**LD INR CLR**

**INPR**

**IR 5**

**LD**

**TR 6**

**LD INR CLR**



**OUTR**

**LD**

**16-bit common bus**

**Clock**

# COMMON BUS SYSTEM

**Memory 4096 x 16**

**Read Write**

**Address**

**E ALU**

**INPR**

**AC**

**L I C**

**PC**

**L I C**

**L I C**

**DR**

**L**

**IR L I C**

**TR**



**AR OUTR LD**

**L I C**

**7 1 2 3 4 5 6**

**16-bit Common Bus**

**S0 S1 S2**

# COMMON BUS SYSTEM

**Register x**

**AR**

**PC DR AC IR TR**

**Memory**

**S2 S1 S0 0 0 0**

**0 0 1**

**0 1 0**

**0 1 1**

**1 0 0**

**1 0 1**

**1 1 0**

**1 1 1**

* **Three control lines, S2, S1, and S0 control which register the bus selects as its input**
* **Either one of the registers will have its load signal activated, or the memory will have its read signal activated**

– **Will determine where the data from the bus gets loaded**

* **The 12-bit registers, AR and PC, have 0’s loaded onto the bus in the high order 4 bit positions**
* **When the 8-bit register OUTR is loaded from the bus, the data comes from the low order 8 bits on the bus**

# BASIC COMPUTER INSTRUCTIONS

* **Basic Computer Instruction Format**

**Memory-Reference Instructions (OP-code = 000 ~ 110)**

**15 14 12 11 0**

**I Opcode Address**

**Register-Reference Instructions (OP-code = 111, I = 0)**

**15 12 11 0**

**0 1 1 1 Register operation**

**Input-Output Instructions (OP-code =111, I = 1)**

**15 12 11 0**

**1 1 1 1**

**I/O operation**

# BASIC COMPUTER INSTRUCTIONS

|  |  |  |
| --- | --- | --- |
| ***Symbol*** | ***Hex Code*** | ***Description*** |
| ***I = 0 I = 1*** |
| **AND ADD LDA STA BUN BSA ISZ** | **0xxx 8xxx**  **1xxx 9xxx**  **2xxx Axxx**  **3xxx Bxxx**  **4xxx Cxxx**  **5xxx Dxxx**  **6xxx Exxx** | **AND memory word to AC Add memory word to AC Load AC from memory**  **Store content of AC into memory Branch unconditionally**  **Branch and save return address Increment and skip if zero** |
| **CLA** | **7800** | **Clear AC** |
| **CLE** | **7400** | **Clear E** |
| **CMA** | **7200** | **Complement AC** |
| **CME** | **7100** | **Complement E** |
| **CIR** | **7080** | **Circulate right AC and E** |
| **CIL** | **7040** | **Circulate left AC and E** |
| **INC** | **7020** | **Increment AC** |
| **SPA** | **7010** | **Skip next instr. if AC is positive** |
| **SNA** | **7008** | **Skip next instr. if AC is negative** |
| **SZA** | **7004** | **Skip next instr. if AC is zero** |
| **SZE** | **7002** | **Skip next instr. if E is zero** |
| **HLT** | **7001** | **Halt computer** |
| **INP** | **F800** | **Input character to AC** |
| **OUT** | **F400** | **Output character from AC** |
| **SKI** | **F200** | **Skip on input flag** |
| **SKO** | **F100** | **Skip on output flag** |
| **ION** | **F080** | **Interrupt on** |
| **IOF** | **F040** | **Interrupt off** |

**INSTRUCTION SET COMPLETENESS**

A computer should have a set of instructions so that the user can construct machine language programs to evaluate any function that is known to be computable.

• Instruction Types

**Functional Instructions**

**- Arithmetic, logic, and shift instructions**

**- ADD, CMA, INC, CIR, CIL, AND, CLA**

**Transfer Instructions**

**- Data transfers between the main memory and the processor registers**

**- LDA, STA**

**Control Instructions**

**- Program sequencing and control**

**- BUN, BSA, ISZ**

**Input/Output Instructions**

**- Input and output**

**- INP, OUT**

# CONTROL UNIT

* **Control unit (CU) of a processor translates from machine instructions to the control signals for the microoperations that implement them**
* **Control units are implemented in one of two ways**
* ***Hardwired* Control**
  + **CU is made up of sequential and combinational circuits to generate the control signals**
* ***Microprogrammed* Control**
  + **A control memory on the processor contains microprograms that activate the necessary control signals**
* **We will consider a hardwired implementation of the control unit for the Basic Computer**

# TIMING AND CONTROL

Control unit of Basic Computer

**Instruction register (IR) 15 14 13 12 11 - 0**

**Other inputs**



**Increment (INR) Clear (CLR)**

**Clock**

**4-bit sequence counter (SC)**

**15 14 . . . . 2 1 0**

**4 x 16**

**decoder**

**T15**

**T0**

**Combinational Control**

**logic**

**7**

**D**

**D0**

**I**

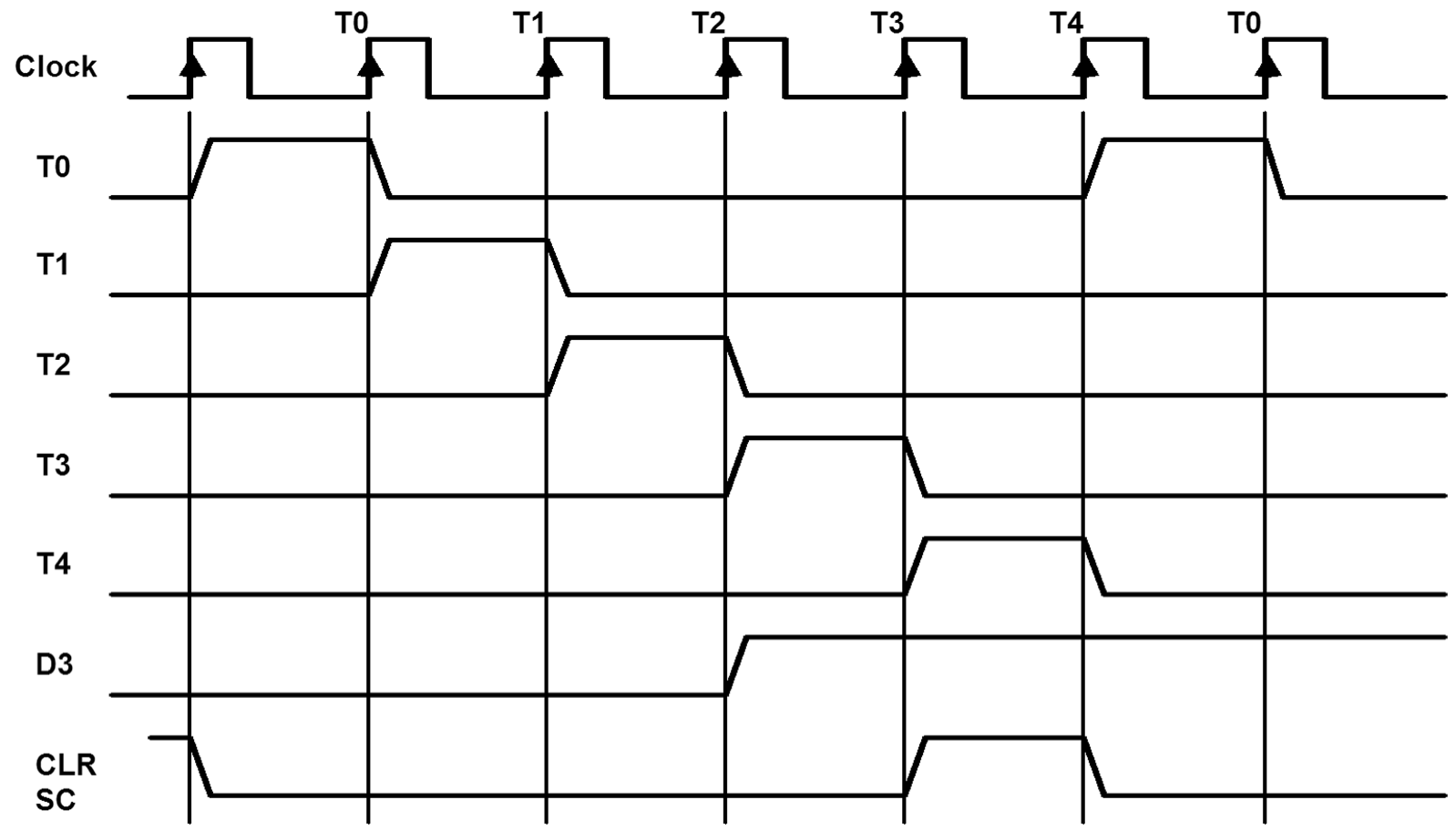
**3 x 8 decoder**

**7 6 5 4 3 2 1 0**

**signals**

**Control**

# TIMING SIGNALS



* **Generated by 4-bit sequence counter and 416 decoder**
* **The SC can be incremented or cleared.**

**- Example: T0, T1, T2, T3, T4, T0, T1, . . .**

**Assume: At time T4, SC is cleared to 0 if decoder output D3 is active.**

**D3T4: SC 0**

# INSTRUCTION CYCLE

• In Basic Computer, a machine instruction is executed in the following cycle:

1. **Fetch an instruction from memory**
2. **Decode the instruction**
3. **Read the effective address from memory if the instruction has an indirect address**
4. **Execute the instruction**

• After an instruction is executed, the cycle starts again at step 1, for the next instruction

• *Note*: Every different processor has its own (different) instruction cycle

# FETCH and DECODE

* **Fetch and Decode**

**T1 T0**

**S2**

**S1 Bus**

**S0**

**Memory unit**

**Read**

**7**

**Address**

**AR 1**

**LD**

**PC 2**



**T0: AR  PC (S0S1S2=010, T0=1)**

**T1: IR  M [AR], PC  PC + 1 (S0S1S2=111, T1=1)**

**T2: D0, . . . , D7  Decode IR(12-14), AR  IR(0-11), I  IR(15)**

**INR**

**IR**

**LD**

**Common bus**

**5**

**Clock**

# DETERMINE THE TYPE OF INSTRUCTION

**Start SC  **

**AR  PC T0**

**T1**

**IR  M[AR], PC  PC + 1**

**T2**

**Decode Opcode in IR(12-14), AR  IR(0-11), I  IR(15)**

**(Register or I/O) = 1**

**= 0 (Memory-reference)**

**D7**

**(I/O) = 1**

**I**

**= 0 (register)**

**(indirect) = 1**

**I**

**= 0 (direct)**

**T3 T3 T3 T3**

**Execute input-output instruction**

**Execute register-reference**

**instruction**

**AR  M[AR]**

**Nothing**

**SC  0**

**SC  0**

**Execute T4 memory-reference**

**instruction**

**SC  0**



**D'7IT3: AR  M[AR]**

**D'7I'T3: Nothing**

**D7I'T3: Execute a register-reference instr. D7IT3: Execute an input-output instr.**

# REGISTER REFERENCE INSTRUCTIONS

**SC  0**

**AC  0**

**E  0**

**AC  AC’ E  E’**

**AC  shr AC, AC(15)  E, E  AC(0) AC  shl AC, AC(0)  E, E  AC(15)**

**AC  AC + 1**

**if (AC(15) = 0) then (PC  PC+1) if (AC(15) = 1) then (PC  PC+1)**

**if (AC = 0) then (PC  PC+1) if (E = 0) then (PC  PC+1)**

**S  0 (S is a start-stop flip-flop)**

**r: rB11: rB10: rB9: rB8: rB7: rB6: rB5: rB4: rB3: rB2: rB1: rB0:**

**CLA CLE CMA CME CIR CIL INC SPA SNA SZA SZE HLT**

**Register Reference Instructions are identified when**

**- D7 = 1, I = 0**

* **Register Ref. Instr. is specified in b0 ~ b11 of IR**
* **Execution starts with timing signal T3**

**r = D7 IT3 => Register Reference Instruction Bi = IR(i) , i=0,1,2,...,11**

# MEMORY REFERENCE INSTRUCTIONS

**Symbol**

**AND ADD LDA STA BUN BSA ISZ**

**Operation Decoder**

**D0 D1 D2 D3 D4 D5 D6**

**Symbolic Description**

**AC  AC  M[AR]**

**AC  AC + M[AR], E  Cout AC  M[AR]**

**M[AR]  AC PC  AR**

**M[AR]  PC, PC  AR + 1**

**M[AR]  M[AR] + 1, if M[AR] + 1 = 0 then PC  PC+1**

* **The effective address of the instruction is in AR and was placed there during timing signal T2 when I = 0, or during timing signal T3 when I = 1**
* **Memory cycle is assumed to be short enough to complete in a CPU cycle**
* **The execution of MR instruction starts with T4**

**AND to AC**

**D0T4: DR  M[AR] Read operand**

**D0T5: AC  AC  DR, SC  0 AND with AC ADD to AC**

**D1T4: DR  M[AR] Read operand**

**D1T5: AC  AC + DR, E  Cout, SC  0 Add to AC and store carry in E**

# MEMORY REFERENCE INSTRUCTIONS

**LDA: Load to AC**

**D2T4: DR  M[AR] D2T5: AC  DR, SC  0**

**STA: Store AC**

**D3T4: M[AR]  AC, SC  0**

**BUN: Branch Unconditionally**

**D4T4: PC  AR, SC  0**

**BSA: Branch and Save Return Address**

**M[AR]  PC, PC  AR + 1**

**Memory, PC, AR at time T4**

**Memory, PC after execution**

**20**

**PC = 21**

**AR = 135**

**136**

**0 BSA 135**

**Next instruction**

**Subroutine**

**20**

**21**

**135**

**PC = 136**

**0 BSA 135**

**Next instruction**

**21**

**Subroutine**



**1 BUN 135 1 BUN 135**

**Memory Memory**

# MEMORY REFERENCE INSTRUCTIONS

**BSA:**

**D5T4: M[AR]  PC, AR  AR + 1 D5T5: PC  AR, SC  0**

**ISZ: Increment and Skip-if-Zero D6T4: DR  M[AR] D6T5: DR  DR + 1**

**D6T4: M[AR]  DR, if (DR = 0) then (PC  PC + 1), SC  0**

## FLOWCHART FOR MEMORY REFERENCE INSTRUCTIONS

**Memory-reference instruction**

**AND ADD LDA STA**

**D0T4 D1T4 D2T4 D3T4 DR  M[AR] DR  M[AR] DR  M[AR] M[AR]  AC**

**SC  0**

**D0T5 AC  AC  DR**

**SC  0**

**D1T5 AC  AC + DR**

**E  Cout SC  0**

**D2T5 AC  DR**

**SC  0**

**BUN BSA ISZ**

**D4T4 D5T4 D6T4**

**PC  AR SC  0**

**M[AR]  PC AR  AR + 1**

**DR  M[AR]**

**PC  AR SC  0**

**D5T5 D6T5**

**DR  DR + 1**

**D6T6**

**M[AR]  DR If (DR = 0)**



**then (PC  PC + 1) SC  0**

# INPUT-OUTPUT AND INTERRUPT

**A Terminal with a keyboard and a Printer**

* **Input-Output Configuration**

**Input-output terminal**

**Printer**

**Serial communication interface**

**Receiver interface**

**Computer registers and flip-flops**

**OUTR**

**FGO**

***INPR* Input register - 8 bits**

**Keyboard**

**Transmitter interface**

**AC**

**INPR FGI**

***OUTR* Output register - 8 bits**

***FGI* Input flag - 1 bit**

***FGO* Output flag - 1 bit**

***IEN* Interrupt enable - 1 bit**

**Serial Communications Path Parallel Communications Path**



* **The terminal sends and receives serial information**
* **The serial info. from the keyboard is shifted into INPR**
* **The serial info. for the printer is stored in the OUTR**
* **INPR and OUTR communicate with the terminal serially and with the AC in parallel.**
* **The flags are needed to *synchronize* the timing difference between I/O device and the computer**

# PROGRAM CONTROLLED DATA TRANSFER

**-- CPU I/O Device --**

**/\* Input \*/ /\* Initially FGI = 0 \*/ loop: If FGI = 0 goto loop**

**AC  INPR, FGI  0**

**/\* Output \*/ /\* Initially FGO = 1 \*/ loop: If FGO = 0 goto loop**

**OUTR  AC, FGO  0**

**FGI=0**

**Start Input FGI  0**

**loop: If FGI = 1 goto loop**

**INPR  new data, FGI  1**

**loop: If FGO = 1 goto loop consume OUTR, FGO  1**

**FGO=1**

**Start Output**

**AC  Data**

**yes**



**yes**

**FGI=0**

**no AC  INPR**

**More Character**

**no END**

**yes**

**yes**

**FGO=0**

**no OUTR  AC**

**FGO  0**

**More Character**

**no END**

# INPUT-OUTPUT INSTRUCTIONS

**Clear SC**

**Input char. to AC Output char. from AC Skip on input flag Skip on output flag Interrupt enable on Interrupt enable off**

**SC  0**

**AC(0-7)  INPR, FGI  0 OUTR  AC(0-7), FGO  0**

**if(FGI = 1) then (PC  PC + 1) if(FGO = 1) then (PC  PC + 1) IEN  1**

**IEN  0**

**p:**

**pB11: pB10: pB9: pB8: pB7:**

**pB6:**

**INP**

**OUT SKI SKO ION IOF**

**D7IT3 = p**

**IR(i) = Bi, i = 6, …, 11**

# PROGRAM-CONTROLLED INPUT/OUTPUT

* + **Program-controlled I/O**
    - **Continuous CPU involvement**

**I/O takes valuable CPU time**

* + - **CPU slowed down to I/O speed**
    - **Simple**
    - **Least hardware**

**Input**

|  |  |  |
| --- | --- | --- |
| **LOOP,** | **SKI** | **DEV** |
|  | **BUN** | **LOOP** |
|  | **INP** | **DEV** |

**Output**

|  |  |  |
| --- | --- | --- |
| **LOOP,** | **LDA** | **DATA** |
| **LOP,** | **SKO** | **DEV** |
|  | **BUN** | **LOP** |
|  | **OUT** | **DEV** |

# INTERRUPT INITIATED INPUT/OUTPUT

* **Open communication only when some data has to be passed --> *interrupt*.**
* **The I/O interface, instead of the CPU, monitors the I/O device.**
* **When the interface founds that the I/O device is ready for data transfer, it generates an interrupt request to the CPU**
* **Upon detecting an interrupt, the CPU stops momentarily the task**

**it is doing, branches to the service routine to process the data transfer, and then returns to the task it was performing.**

* **IEN (Interrupt-enable flip-flop)**
  + **can be set and cleared by instructions**
  + **when cleared, the computer cannot be interrupted**

# FLOWCHART FOR INTERRUPT CYCLE

**R = Interrupt f/f**

**Instruction cycle**

**Fetch and decode instructions**

**=0 R =1**

**Interrupt cycle**

**Store return address in location 0**

**M[0]  PC**

**Execute instructions**

**IEN =0**

**=1**

**=1 FGI**

**=0**

**=1 FGO**

**=0**

**Branch to location 1 PC  1**

**IEN  0**

**R  0**

**R  1**



* **The interrupt cycle is a HW implementation of a branch and save return address operation.**
* **At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1.**
* **At memory address 1, the programmer must store a branch instruction that sends the control to an interrupt service routine**
* **The instruction that returns the control to the original program is "indirect BUN 0"**

REGISTER TRANSFER OPERATIONS IN INTERRUPT CYCLE

**Before interrupt**

**0**

**Memory**

**After interrupt cycle 0 256**

**1 0 BUN 1120**

**PC = 1**

**0 BUN 1120**

**255**

**PC = 256**

**1120**

**Main Program**

**I/O Program**

**255**

**256**

**1120**

**Main Program**

**I/O Program**



**1 BUN 0 1 BUN 0**

**Register Transfer Statements for Interrupt Cycle**

**- R F/F  1 if IEN (FGI + FGO)T0T1T2**

** T0T1T2 (IEN)(FGI + FGO): R  1**

* + **The fetch and decode phases of the instruction cycle**

**must be modified Replace T0, T1, T2 with R'T0, R'T1, R'T2**

* + **The interrupt cycle :**

**RT0: AR  0, TR  PC RT1: M[AR]  TR, PC  0**

**RT2: PC  PC + 1, IEN  0, R  0, SC  0**

# FURTHER QUESTIONS ON INTERRUPT

How can the CPU recognize the device requesting an interrupt ?

Since different devices are likely to require different interrupt service routines, how can the CPU obtain the starting address of the appropriate routine in each case ?

Should any device be allowed to interrupt the CPU while another interrupt is being serviced ?

How can the situation be handled when two or more interrupt requests occur simultaneously ?

# COMPLETE COMPUTER DESCRIPTION

Flowchart of Operations

**start**

**SC  0, IEN  0, R  0**

**=0(Instruction R =1(Interrupt**

**Cycle) Cycle)**

**R’T0**

**RT0**

**AR  PC**

**R’T1**

**AR  0, TR  PC**

**RT1**

**IR  M[AR], PC  PC + 1**

**R’T2**

**M[AR]  TR, PC  0**

**RT2**

**AR  IR(0~11), I  IR(15)**

**D0...D7  Decode IR(12 ~ 14)**

**PC  PC + 1, IEN  0 R  0, SC  0**

**=1(Register or I/O)**

**D7 =0(Memory Ref)**

**=1 (I/O) I**

**=0 (Register) =1(Indir)**

**I**

**=0(Dir)**

**D7IT3 D7I’T3 D7’IT3 D7’I’T3**

**Execute I/O Instruction**



**Execute RR**

**Instruction**

**AR <- M[AR] Idle**

**Execute MR Instruction**

**D7’T4**

# COMPLETE COMPUTER DESCRIPTION

Microoperations

**Fetch Decode**

**Indirect Interrupt**

**RT0: RT1: RT2:**

**D7IT3:**

**AR  PC**

**IR  M[AR], PC  PC + 1**

**D0, ..., D7  Decode IR(12 ~ 14), AR  IR(0 ~ 11), I  IR(15)**

**AR  M[AR]**

**T0T1T2(IEN)(FGI + FGO):**

**RT0: RT1: RT2:**

**Memory-Reference**

**R  1**

**AR  0, TR  PC M[AR]  TR, PC  0**

**PC  PC + 1, IEN  0, R  0, SC  0**

**AND ADD LDA**

**STA BUN BSA**

**ISZ**

**D0T4: D0T5: D1T4: D1T5: D2T4: D2T5: D3T4: D4T4: D5T4: D5T5: D6T4: D6T5: D6T6:**

**DR  M[AR]**

**AC  AC  DR, SC  0 DR  M[AR]**

**AC  AC + DR, E  Cout, SC  0 DR  M[AR]**

**AC  DR, SC  0 M[AR]  AC, SC  0 PC  AR, SC  0**

**M[AR]  PC, AR  AR + 1 PC  AR, SC  0**

**DR  M[AR] DR  DR + 1**

**M[AR]  DR, if(DR=0) then (PC  PC + 1), SC  0**

# COMPLETE COMPUTER DESCRIPTION

Microoperations

**Register-Reference**

**D7IT3 = r IR(i) = Bi**

**r:**

**(Common to all register-reference instr) (i = 0,1,2, ..., 11)**

**SC  0**

**CLA CLE CMA CME CIR CIL INC SPA SNA SZA SZE HLT**

**Input-Output**

**INP OUT SKI SKO ION IOF**

**rB11: rB10: rB9: rB8: rB7: rB6: rB5: rB4: rB3: rB2: rB1: rB0:**

**D7IT3 = p IR(i) = Bi**

**p:**

**pB11: pB10: pB9: pB8: pB7: pB6:**

**AC  0**

**E  0**

**AC  AC E  E**

**AC  shr AC, AC(15)  E, E  AC(0) AC  shl AC, AC(0)  E, E  AC(15) AC  AC + 1**

**If(AC(15) =0) then (PC  PC + 1) If(AC(15) =1) then (PC  PC + 1) If(AC = 0) then (PC  PC + 1) If(E=0) then (PC  PC + 1)**

**S  0**

**(Common to all input-output instructions) (i = 6,7,8,9,10,11)**

**SC  0**

**AC(0-7)  INPR, FGI  0 OUTR  AC(0-7), FGO  0 If(FGI=1) then (PC  PC + 1) If(FGO=1) then (PC  PC + 1) IEN  1**

**IEN  0**