Register Transfer Language and Basic Computer Organization

Computer Architecture — Module 3 Notes

1. Register Transfer Language (RTL)

1.1 Introduction to RTL

Register Transfer Language (RTL) is a symbolic notation used to describe the micro-operations and data transfers between registers in a digital system.

Key Concepts:

- Registers: Storage locations in CPU
- Micro-operations: Elementary operations performed on data in registers
- Register Transfer: Moving data from one register to another

1.2 Notation in RTL

Symbol	Meaning	Example
R0, R1, R2...	Register names	R1, R2
←	Transfer (assignment)	R1 ← R2
[ ]	Contents of register/memory	[R1]
:	Condition	P: R1 ← R2
,	Simultaneous operations	R1 ← R2, R3 ← R4
( )	Part of register	R1(0-7)
+	Addition	R1 ← R2 + R3
-	Subtraction	R1 ← R2 - R3
&	AND	R1 ← R2 & R3
OR	OR	R1 ← R2 OR R3
⊕	XOR	R1 ← R2 ⊕ R3

1.3 Basic Register Transfer Operations

Direct Transfer:

```
R1 ← R2    ; Transfer contents of R2 to R1
```

Conditional Transfer:

```
P: R1 ← R2    ; If P=1, transfer R2 to R1
```

Simultaneous Transfers:

```
R1 ← R2, R3 ← R4    ; Both transfers occur simultaneously
```

Example Scenarios

1. Simple Data Movement:

```
R1 ← R2    ; R1 gets value from R2
```

2. Arithmetic Operation:

```
R1 ← R2 + R3    ; Add R2 and R3, store in R1
```

3. Conditional Operation:

```
P: R1 ← R2    ; If condition P is true
Q: R1 ← R3    ; If condition Q is true
```

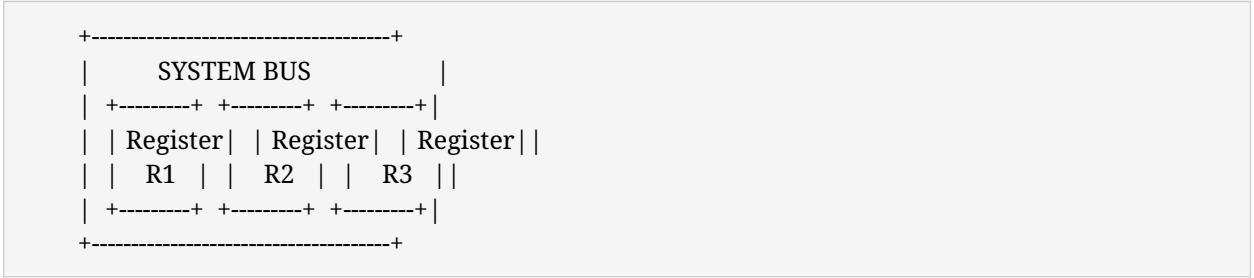
2. Bus and Memory Transfers

2.1 Bus System Architecture

A bus is a set of common lines that connect multiple registers.

Types of Buses:

- Data Bus: Carries data
- Address Bus: Carries addresses
- Control Bus: Carries control signals



2.2 Three-State Bus Buffers

Three-State Buffer: Can be in one of three states:

- Logic 0
- Logic 1
- High Impedance (Hi-Z) — Disconnected from bus

```
      +-------+
Data ---|       |--- Bus
      | Tri-  |
Enable--| State |
      | Buffer|
      +-------+

Enable = 1: Data passes through
Enable = 0: Output is Hi-Z (disconnected)
```

Advantages:

- Multiple devices can share bus
- Only active device drives bus
- Prevents bus conflicts

2.3 Bus Transfer with Tri-State Buffers

```
Control Signals:
S0, S1, S2, ... Select which register drives the bus

For n registers:
- Need n control signals
- Only one control signal active at a time
```

Example: 4-Register Bus System

```
Registers: R0, R1, R2, R3
Control: E0, E1, E2, E3

When E0 = 1: R0 drives the bus
When E1 = 1: R1 drives the bus
When E2 = 1: R2 drives the bus
When E3 = 1: R3 drives the bus
```

2.4 Memory Transfer Operations

Memory Read:

```
R1 ← M[Address]    ; Read from memory to register
```

Memory Write:

```
M[Address] ← R1    ; Write from register to memory
```

Timing in Memory Transfers:

```
Address Setup:     |=== Address ===|
Memory Enable:     |=== MEM Enable ===|
Data Read/Write:   |=== Data ===|
                   ^
                   Clock Edge
```

2.5 Memory Access Timing

Read Cycle:

- Place address on address bus
- Assert memory read signal (MEMR)
- Wait for memory access time
- Data appears on data bus
- Capture data in register
- Deassert signals

Write Cycle:

- Place address on address bus
- Place data on data bus
- Assert memory write signal (MEMW)
- Memory stores data
- Deassert signals

3. Micro-Operations

3.1 Types of Micro-operations

Type	Description	Example
Register Transfer	Transfer data between registers	R1 ← R2
Arithmetic	Arithmetic operations	R1 ← R2 + R3
Logic	Logical operations	R1 ← R2 AND R3

Type	Description	Example
Shift	Shift operations	R1 ← SHL(R2)

3.2 Arithmetic Micro-operations

Operation	Symbol	Description
Addition	+	R1 ← R2 + R3
Subtraction	-	R1 ← R2 - R3
Increment	+1	R1 ← R1 + 1
Decrement	-1	R1 ← R1 - 1
Add with Carry	+	R1 ← R2 + R3 + Cin
Subtract with Borrow	-	R1 ← R2 - R3 - Bin
2's Complement	-	R1 ← R2' + 1

Example:

R1 ← R1 + R2
R2 ← R2 - 1
R3 ← R3 + 1 (if carry)

3.3 Logic Micro-operations

Operation	Symbol	Description
OR	OR	R1 ← R2 OR R3
AND	&	R1 ← R2 & R3
XOR	⊕	R1 ← R2 ⊕ R3
Complement	'	R1 ← R2'
NAND	↑	R1 ← R2 ↑ R3
NOR	↓	R1 ← R2 ↓ R3
XNOR	⊙	R1 ← R2 ⊙ R3

Applications:

- OR: Setting bits
- AND: Clearing/Testing bits
- XOR: Complementing bits

3.4 Shift Micro-operations

Types of Shifts:

Type	Description	Example
Logical Shift Left	Shifts left, zeros fill right	1101 → 1010
Logical Shift Right	Shifts right, zeros fill left	1101 → 0110
Arithmetic Shift Left	Same as logical, sign preserved	1101 → 1010
Arithmetic Shift Right	Sign bit shifted in	1101 → 1110
Rotate Left	Bits wrap around	1101 → 1011
Rotate Right	Bits wrap around	1101 → 1110
Rotate with Carry	Includes carry bit	1101 → carry + bits

Notation:

```
SHL: Shift Left
SHR: Shift Right
ROR: Rotate Right
ROL: Rotate Left
```

3.5 Arithmetic-Logic Shift Unit (ALSU)

ALU (Arithmetic Logic Unit): Performs arithmetic and logic operations.

Components:

- Arithmetic Unit: Addition, subtraction, increment, decrement
- Logic Unit: AND, OR, XOR, NOT
- Shift Unit: Shift, rotate operations
- Control Unit: Selects operation

Control Signals in ALU:

```
S0, S1: Select operation
Cin: Carry input
Cout: Carry output
V: Overflow flag
Z: Zero flag
N: Negative flag
```

ALU Truth Table Example:

S1	S0	Cin	Operation
0	0	0	R1 ← R2 + R3

S1	S0	Cin	Operation
0	0	1	R1 ← R2 + R3 + 1
0	1	0	R1 ← R2 - R3
0	1	1	R1 ← R2 - R3 - 1
1	0	0	R1 ← R2 AND R3
1	0	1	R1 ← R2 OR R3
1	1	0	R1 ← R2 XOR R3
1	1	1	R1 ← R2'

4. Instruction Codes and Computer Registers

4.1 Instruction Code Format

Instruction Format:

```
+----------------+------------------+
|  Opcode        |  Operand/Address |
+----------------+------------------+
|  8 bits        |  8 bits          |
```

Parts of Instruction:

- Opcode: Operation to perform
- Operand: Data or address to operate on
- Address: Memory location of data

Instruction Types:

- Memory Reference: Operates on memory data
- Register Reference: Operates on register data
- Input/Output: I/O operations

4.2 Common Registers in Basic Computer

Register	Symbol	Size	Purpose
Program Counter	PC	12-bit	Holds address of next instruction
Memory Address Register	MAR	12-bit	Holds address for memory access
Memory Buffer Register	MBR	16-bit	Holds data from/to memory

Register	Symbol	Size	Purpose
Accumulator	AC	16-bit	Stores ALU results
Instruction Register	IR	16-bit	Holds current instruction
Data Register	DR	16-bit	Holds data for operations
Temporary Register	TR	16-bit	Temporary storage
Input/Output Register	INPR/OUTR	8-bit	I/O data storage

4.3 Register Functions

Program Counter (PC):

```
PC ← PC + 1    ; Increment for next instruction
PC ← Address   ; Jump to new address
```

Memory Address Register (MAR):

```
MAR ← PC      ; For instruction fetch
MAR ← Address ; For data access
```

Memory Buffer Register (MBR):

```
MBR ← M[MAR]  ; Read from memory
M[MAR] ← MBR  ; Write to memory
```

Accumulator (AC):

```
AC ← AC + DR  ; Add
AC ← AC & DR  ; AND
AC ← AC'      ; Complement
```

4.4 Register Transfer Notation Examples

Instruction Fetch:

```
T0: MAR ← PC
T1: MBR ← M[MAR]
T2: IR ← MBR
T3: PC ← PC + 1
```

Execution of ADD Instruction:

```
T0: MAR ← IR(0-11)    ; Get address from instruction
```

```
T1: MBR ← M[MAR]     ; Read operand
T2: AC ← AC + MBR    ; Add to accumulator
```

Execution of AND Instruction:

```
T0: MAR ← IR(0-11)
T1: MBR ← M[MAR]
T2: AC ← AC & MBR
```

5. Computer Instructions

5.1 Types of Instructions

Type	Format	Description
Memory Reference	Opcode(3) Address(12)	Uses memory address operand
Register Reference	Opcode(3) Register(12)	Uses register operand
Input/Output	Opcode(3) I/O(12)	I/O operations

5.2 Memory Reference Instructions

Instruction	Format	Operation
AND	AND Address	AC ← AC AND M[Address]
ADD	ADD Address	AC ← AC + M[Address]
LDA (Load Accumulator)	LDA Address	AC ← M[Address]
STA (Store Accumulator)	STA Address	M[Address] ← AC
BUN (Branch Unconditionally)	BUN Address	PC ← Address
BSA (Branch and Save Return Address)	BSA Address	M[Address] ← PC+1, PC ← Address+1
ISZ (Increment and Skip if Zero)	ISZ Address	M[Address] ← M[Address] + 1; If M[Address] = 0 then PC ← PC + 1

5.3 Register Reference Instructions

Instruction	Symbol	Operation
Clear Accumulator	CLA	AC ← 0
Clear AC	CLE	No operation
Complement Accumulator	CMA	AC ← AC'
Complement E	CME	E ← E'

Instruction	Symbol	Operation
Circulate E and AC	CIR	Rotate right AC(15) ↔ E
Circulate E and AC	CIL	Rotate left AC(15) ↔ E
Increment AC	INC	AC ← AC + 1
Skip if AC=0	SPA	If AC>0 then skip
Skip if AC=0	SNA	If AC<0 then skip
Skip if E=0	SZA	If AC=0 then skip
Skip if E=1	SZE	If E=0 then skip
Halt	HLT	Stop execution

5.4 Input/Output Instructions

Instruction	Symbol	Operation
Input Character	INP	AC(0-7) ← INPR
Output Character	OUT	OUTR ← AC(0-7)
Skip if Input Flag	SKI	If FGI=1 then skip
Skip if Output Flag	SKO	If FGO=1 then skip
Interrupt On	ION	IEN ← 1
Interrupt Off	IOF	IEN ← 0

6. Timing and Control

6.1 Timing and Control Signals

Control Unit: Generates timing and control signals for instruction execution.

Timing Signals:

- Generated by a clock and counter
- T0, T1, T2, T3, ... (time states)

Control Signals:

- Command Signals: Initiate operations in various units
- Enable Signals: Enable or disable operations
- Select Signals: Choose between operations

```
      +-----------+
CLK ----|           |
```

```
      |  Control  |---- Control Signals
RST ----|  Unit     |
      |           |
      +-----------+
```

6.2 Instruction Cycle

The instruction cycle consists of:

1. Instruction Fetch (IF):

```
T0: MAR ← PC
T1: MBR ← M[MAR]
T2: IR ← MBR
T3: PC ← PC + 1
```

2. Instruction Decode (ID):

```
T0: Decode IR(12-14) to determine instruction type
```

3. Instruction Execute:

- Based on decoded instruction

6.3 Timing for Different Instructions

Direct Memory Reference:

```
T0: MAR ← IR(0-11)
T1: MBR ← M[MAR]    ; Read operand
T2: AC ← AC + MBR   ; Execute operation
```

Indirect Memory Reference:

```
T0: MAR ← IR(0-11)
T1: MBR ← M[MAR]    ; Read address
T2: MAR ← MBR       ; Load effective address
T3: MBR ← M[MAR]    ; Read operand
T4: AC ← AC + MBR   ; Execute operation
```

6.4 Control Signal Generation

Hardwired Control:

- Uses logic gates

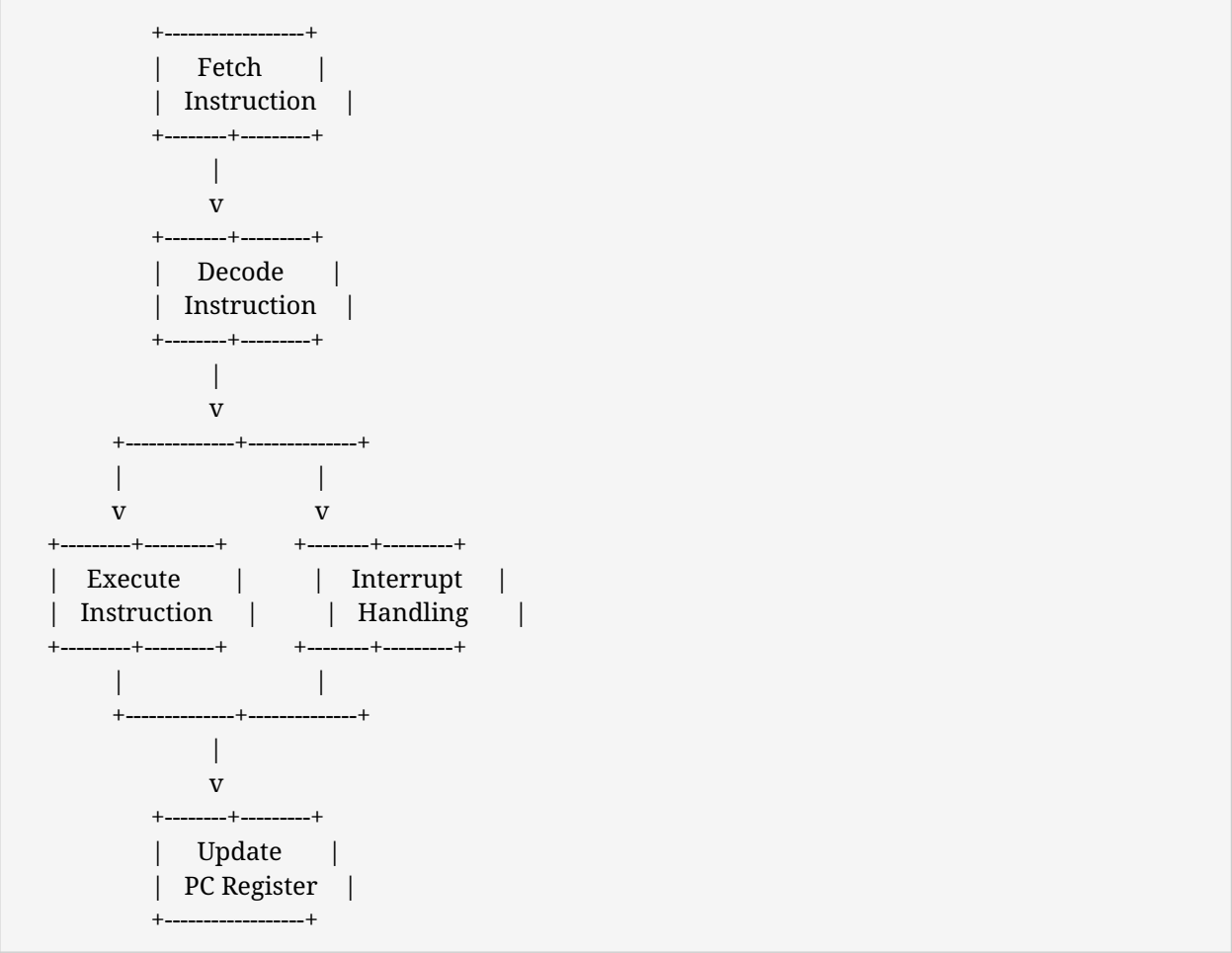
- Fast, but complex for complex instructions
- Used in RISC processors

Microprogrammed Control:

- Uses microcode stored in ROM
- Simple to modify
- Used in CISC processors

7. Instruction Cycle and Interrupts

7.1 Complete Instruction Cycle



7.2 Interrupts

Definition: Signal that temporarily halts normal program execution to handle a specific event.

Types:

- Hardware Interrupts: From external devices

- Software Interrupts: From program instructions
- Internal Interrupts: From CPU exceptions

Interrupt Handling Process:

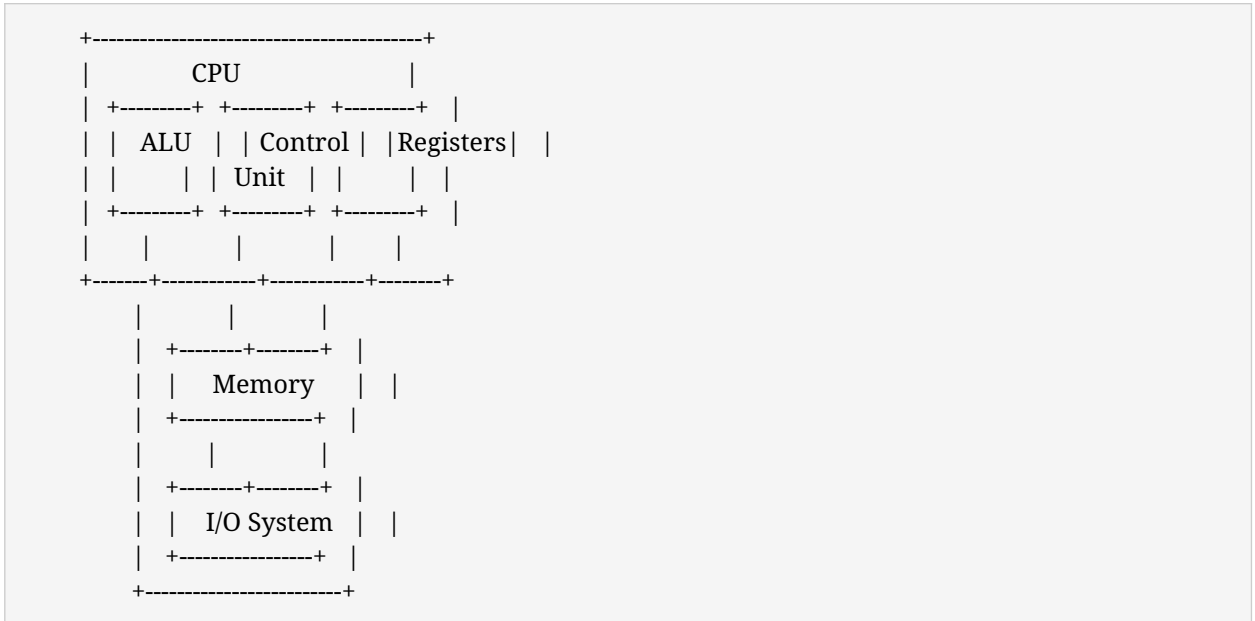
```
1. Complete current instruction
2. Save PC and processor state
3. Disable interrupts
4. Jump to interrupt service routine (ISR)
5. Execute ISR
6. Restore state
7. Return to interrupted program
```

Interrupt Cycle:

```
T0: MBR ← PC
T1: MAR ← Address
T2: M[MAR] ← MBR
T3: PC ← Interrupt Vector Address
T4: Enable interrupts
```

8. Design of Basic Computer

8.1 Basic Computer Architecture



8.2 Data Paths

Main Data Paths:

```
Memory ←→ MBR ←→ AC ←→ ALU
  ↑       ↑       ↑
  |       |       |
 PC      IR      Registers
  |       |       |
  +---Address Bus---+
```

8.3 Control Flow

Instruction Fetch:

```
1. PC → MAR
2. M[MAR] → MBR
3. MBR → IR
4. PC ← PC + 1
```

Instruction Execute:

```
1. Decode IR
2. Generate control signals
3. Execute operation
4. Store result
```

9. Design of Accumulator Unit

9.1 Accumulator Structure

```
+-----------------------------------------+
|          Accumulator Unit               |
|                                         |
| +---------+  +---------+  +---------+   |
| |  AND   | |   OR   | |  XOR   |   |
| +---------+  +---------+  +---------+   |
|     |          |          |       |
|     +------------+------------+       |
|              |                 |
|          +-----+-----+            |
|          |  MUX/ALU |            |
|          +-----+-----+            |
|              |                 |
|          +-----+-----+            |
|          |  Register |            |
|          |   AC    |            |
|          +-----+-----+            |
|              |                 |
|          +-----+-----+            |
```

```
|          | Output  |           |
|          | Buffer  |           |
|          +---------+           |
+----------------------------------------+
```

9.2 Accumulator Operations

Load:

```
AC ← Data
```

Add:

```
AC ← AC + Data
```

Store:

```
M[Address] ← AC
```

Logic Operations:

```
AC ← AC AND Data
AC ← AC OR Data
AC ← AC XOR Data
```

Shift Operations:

```
AC ← SHL(AC)
AC ← SHR(AC)
AC ← ROL(AC)
AC ← ROR(AC)
```

9.3 Accumulator Control Signals

Signal	Function
LD_AC	Load AC with data
ADD_AC	Add data to AC
AND_AC	AND data with AC
COM_AC	Complement AC
SHL_AC	Shift AC left
SHR_AC	Shift AC right

Signal	Function
CLR_AC	Clear AC

10. Important Formulas and Diagrams

10.1 RTL Notation Summary

Operation	RTL Expression
Transfer	R1 ← R2
Addition	R1 ← R2 + R3
Subtraction	R1 ← R2 - R3
AND	R1 ← R2 & R3
OR	R1 ← R2 OR R3
XOR	R1 ← R2 ⊕ R3
Complement	R1 ← R2'
Increment	R1 ← R1 + 1
Decrement	R1 ← R1 - 1

10.2 Memory Access Formulas

```
Read: M[Address] → Data
Write: Data → M[Address]

MAR = Address register
MBR = Buffer register

Read: MBR ← M[MAR]
Write: M[MAR] ← MBR
```

10.3 Instruction Formats

Memory Reference:

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
+--+--+--+-----------------------------+
| Opcode |       Address               |
| 3-bit  |         12-bit              |
+--------+-----------------------------+
```

Register Reference:

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
+--+--+--+-----------------------------+
| 1 1 1 | Register Reference Code      |
|       |         13-bit               |
+-------+-----------------------------+
```

Input/Output:

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
+--+--+--+-----------------------------+
| 1 1 1 | I/O Operation Code           |
|       |         13-bit               |
+-------+-----------------------------+
```

11. Practice Problems

Problem 1: RTL Expression

Given: R1, R2, R3 are 8-bit registers. Write RTL for:

- Transfer R2 to R1
- Add R3 to R2, store in R1
- AND R2 and R3, store in R1
- Complement R2, store in R1

Solution:

```
1. R1 ← R2
2. R1 ← R2 + R3
3. R1 ← R2 & R3
4. R1 ← R2'
```

Problem 2: Memory Transfer

Show the sequence for:

```
M[200] ← R1
```

Solution:

```
Step 1: MAR ← 200
Step 2: MBR ← R1
Step 3: M[MAR] ← MBR
```

Problem 3: Instruction Execution

Trace execution of:

```
ADD 200
Assuming: AC = 5, M[200] = 7
```

Solution:

```
T0: MAR ← 200
T1: MBR ← M[200] = 7
T2: AC ← AC + MBR = 5 + 7 = 12
Result: AC = 12
```

Problem 4: Interrupt Handling

Sequence for handling interrupt:

```
1. Save PC
2. Jump to ISR
3. Execute ISR
4. Restore PC
5. Continue execution
```

12. Key Takeaways

✅ RTL provides formal notation for describing data transfers

✅ Bus system allows multiple registers to share data path

✅ Three-state buffers enable bus sharing without conflicts

✅ Micro-operations are fundamental operations on data

✅ ALU performs arithmetic, logic, and shift operations

✅ Instruction cycle consists of fetch, decode, and execute

✅ Registers store data, addresses, and instruction codes

✅ Interrupts allow CPU to respond to external events

✅ Timing and control coordinate all operations

✅ Accumulator is the primary register for arithmetic operations

These comprehensive notes cover Module 3 of Computer Architecture. Focus on understanding RTL notation, bus transfers, micro-operations, and the instruction cycle. Practice writing RTL expressions and tracing instruction execution.