MODULE 3: BASIC PROCESSING UNIT

SOME FUNDAMENTAL CONCEPTS

• To execute an instruction, processor has to perform following 3 steps:

1) Fetch contents of memory-location pointed to by PC. Content of this location is an instruction to be executed. The instructions are loaded into IR, Symbolically, this operation is written as:

IR [[PC]]

2) Increment PC by 4.

PC [PC] +4

3) Carry out the actions specified by instruction (in the IR).

• The first 2 steps are referred to as **Fetch Phase**.

Step 3 is referred to as **Execution Phase**.

• The operation specified by an instruction can be carried out by performing one or more of the following actions:

1) Read the contents of a given memory-location and load them into a register.

2) Read data from one or more registers.

3) Perform an arithmetic or logic operation and place the result into a register.

4) Store data from a register into a given memory-location.

• The hardware-components needed to perform these actions are shown in Figure3.1

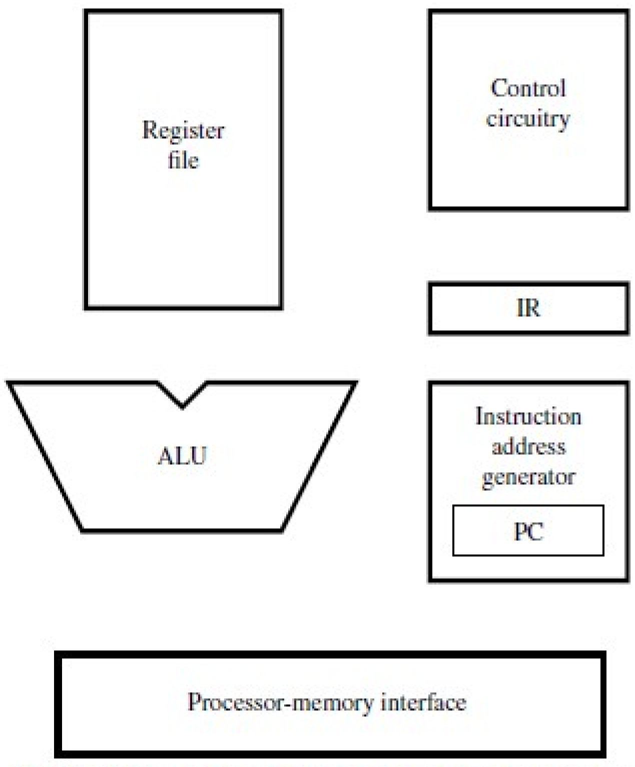


Figure 3.1 main hardware components of a processor

SINGLE BUS ORGANIZATION

• ALU and all the registers are interconnected via a **Single Common Bus** (Figure 3.2).

• Data & address lines of the external memory-bus is connected to the internal processor-bus via MDR

& MAR respectively. (MDR Memory Data Register, MAR Memory Address Register).

• **MDR** has 2 inputs and 2 outputs. Data may be loaded

→ into MDR either from memory-bus (external) or

→ from processor-bus (internal).

• **MAR**"s input is connected to internal-bus;

MAR"s output is connected to external-bus.

• **Instruction Decoder & Control Unit** is responsible for

→ issuing the control-signals to all the units inside the processor.

→ implementing the actions specified by the instruction (loaded in the IR).

• Register R0 through R(n-1) are the **Processor Registers**.

The programmer can access these registers for general-purpose use.

• Only processor can access 3 registers **Y**, **Z** & **Temp** for temporary storage during program-execution.

The programmer cannot access these 3 registers.

• In **ALU**, 1) „A" input gets the operand from the output of the multiplexer (MUX).

2) „B" input gets the operand directly from the processor-bus.

• There are 2 options provided for „A" input of the ALU.

• MUX is used to select one of the 2 inputs.

• **MUX** selects either

→ output of Y or

→ constant-value 4(which is used to increment PC content).

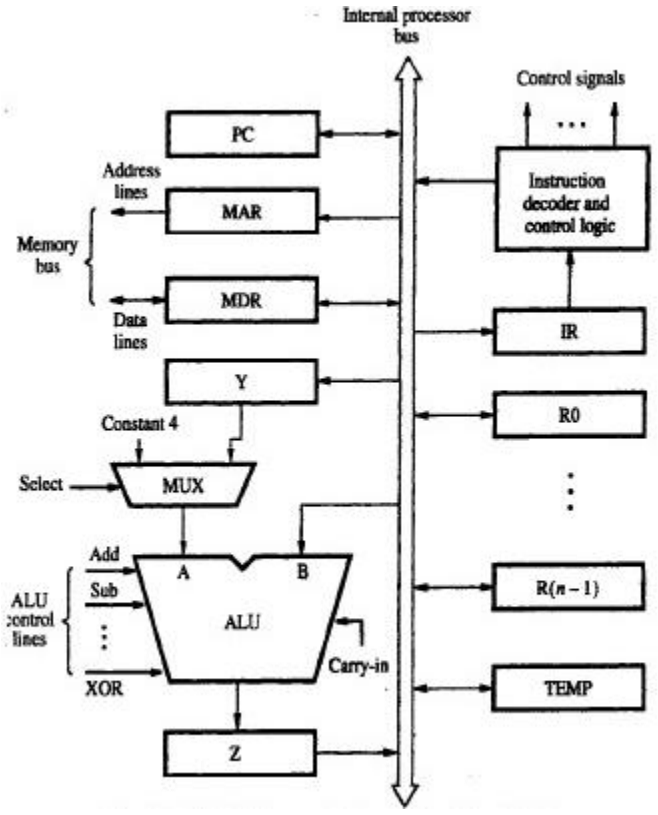


Figure 3.2 Single bus organizations

• An instruction is executed by performing one or more of the following operations:

1) Transfer a word of data from one register to another or to the ALU.

2) Perform arithmetic or a logic operation and store the result in a register.

3) Fetch the contents of a given memory-location and load them into a register.

4) Store a word of data from a register into a given memory-location.

• **Disadvantage:** Only one data-word can be transferred over the bus in a clock cycle.

Solution: Provide multiple internal-paths. Multiple paths allow several data-transfers to take place in parallel.

REGISTER TRANSFERS

• Instruction execution involves a sequence of steps in which data are transferred from one register to another.

• For each register, two control-signals are used: Ri_{in} & Ri_{out}. These are called **Gating Signals.**

• $Ri_{in}=1$ data on bus is loaded into Ri.

$Ri_{out}=1$ content of Ri is placed on bus.

$Ri_{out}=0$, bus can be used for transferring data from other registers.

• For example, *Move R1, R2;* This transfers the contents of register R1 to register R2. This can be

accomplished as follows:

1) Enable the output of registers R1 by setting $R1_{out}$ to 1 (Figure 3.3).

This places the contents of R1 on processor-bus.

2) Enable the input of register R2 by setting $R2_{out}$ to 1.

This loads data from processor-bus into register R4.

• All operations and data transfers within the processor take place within time-periods defined by the

processor-clock.

• The control-signals that govern a particular transfer are asserted at the start of the clock cycle.

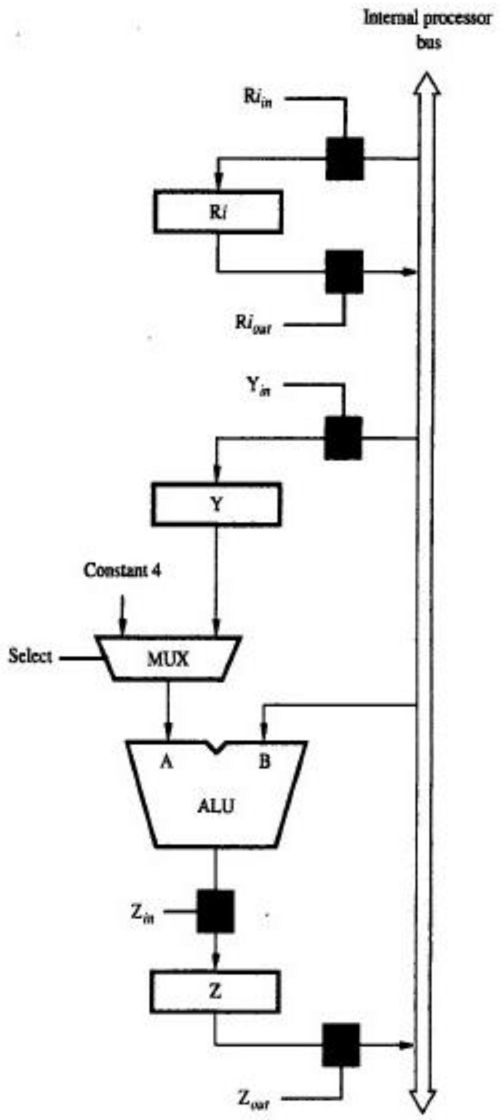


Figure 3.3 input and output gating for the register in figure 3.2

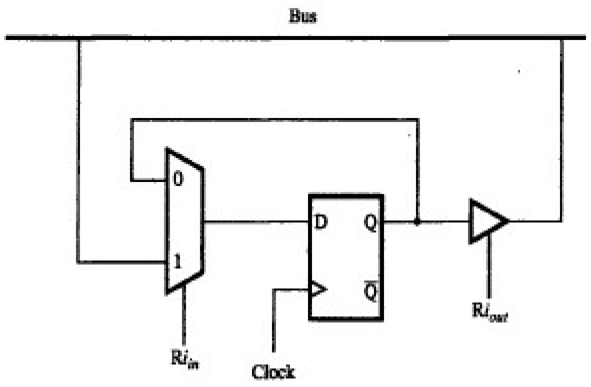


Figure 3.4 input and output gating for one register

Input & Output Gating for one Register Bit

• A 2-input multiplexer is used to select the data applied to the input of an edge-triggered D flip-flop.

• Ri_{in}=1 mux selects data on bus. This data will be loaded into flip-flop at rising-edge of clock.

Ri_{in}=0 mux feeds back the value currently stored in flip-flop (Figure 7.3).

• Q output of flip-flop is connected to bus via a tri-state gate.

Ri_{out}=0 gate's output is in the high-impedance state.

Ri_{out}=1 the gate drives the bus to 0 or 1, depending on the value of Q.

PERFORMING AN ARITHMETIC OR LOGIC OPERATION

• The ALU performs arithmetic operations on the 2 operands applied to its A and B inputs.

• One of the operands is output of MUX;

And, the other operand is obtained directly from processor-bus.

• The result (produced by the ALU) is stored temporarily in register Z.

• The sequence of operations for [R3][R1]+[R2] is as follows:

1) $R1_{out}$, Y_{in}

2) $R2_{out}$, SelectY, Add, Z_{in}

3) Z_{out}, $R3_{in}$

• Instruction execution proceeds as follows:

Step 1 --> Contents from register R1 are loaded into register Y.

Step2 --> Contents from Y and from register R2 are applied to the A and B inputs of ALU; Addition is performed &

Result is stored in the Z register.

Step 3 --> The contents of Z register is stored in the R3 register.

• The signals are activated for the duration of the clock cycle corresponding to that step. All other signals are inactive.

CONTROL-SIGNALS OF MDR

• The MDR register has 4 control-signals (Figure 3.4):

1) MDR_{in} & MDR_{out} control the connection to the internal processor data bus &

2) MDR_{inE} & MDR_{outE} control the connection to the memory Data bus.

• MAR register has 2 control-signals.

1) MAR_{in} controls the connection to the internal processor address bus &

2) MAR_{out} controls the connection to the memory address bus.

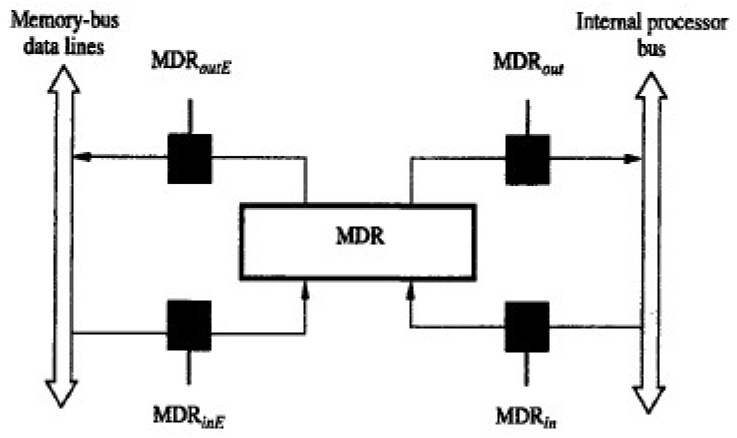


Figure 3.4 Connection and control signal for register MDR.

FETCHING A WORD FROM MEMORY

• To fetch instruction/data from memory, processor transfers required address to MAR.

At the same time, processor issues Read signal on control-lines of memory-bus.

• When requested-data are received from memory, they are stored in MDR. From MDR, they are transferred to other registers.

• The response time of each memory access varies (based on cache miss, memory-mapped I/O). To accommodate this, MFC is used. (MFC Memory Function Completed).

• MFC is a signal sent from addressed-device to the processor. MFC informs the processor that the requested operation has been completed by addressed-device.

• Consider the instruction Move (R1),R2. The sequence of steps is (Figure 7.5):

1) $R1_{out}$, MAR_{in}, Read ;desired address is loaded into MAR & Read command is issued.

2) MDR_{inE}, WMFC ;load MDR from memory-bus & Wait for MFC response from memory.

3) MDR_{out}, $R2_{in}$;load R2 from MDR.

where WMFC=control-signal that causes processor's control.

circuitry to wait for arrival of MFC signal.

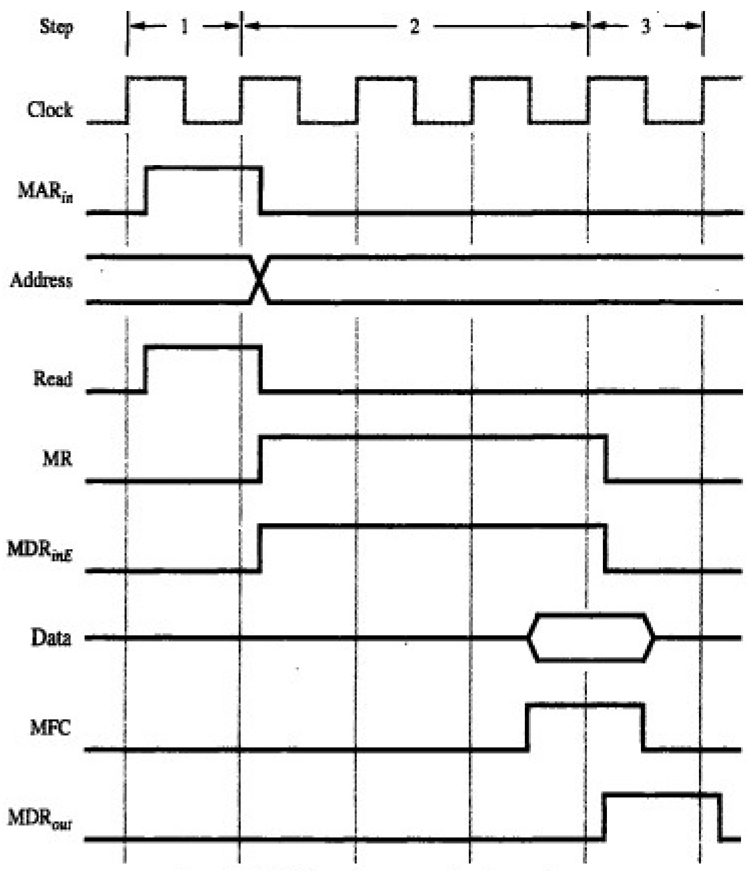


Figure 3.5 Timing of memory read operation

Storing a Word in Memory

• Consider the instruction *Move R2,(R1).* This requires the following sequence:

1) $R1_{out}$, MAR_{in} ;desired address is loaded into MAR.

2) $R2_{out}$, MDR_{in}, Write ;data to be written are loaded into MDR & Write command is issued.

3) MDR_{outE}, WMFC ;load data into memory-location pointed by R1 from MDR.

EXECUTION OF A COMPLETE INSTRUCTION

• Consider the instruction *Add (R3),R1* which adds the contents of a memory-location pointed by R3 to register R1. Executing this instruction requires the following actions:

1) Fetch the instruction.

2) Fetch the first operand.

3) Perform the addition &

4) Load the result into R1.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

Figure 3.6 Control sequence for execution of the instruction Add (r3),R1

• Instruction execution proceeds as follows:

Step1--> The instruction-fetch operation is initiated by

→ loading contents
of PC into MAR &

→ sending a Read
request to memory.

The Select signal is set to Select4, which causes the Mux to select constant 4. This value

is added to operand at input B (PC"s content), and the result is stored in Z.

Step2--> Updated value in Z is moved to PC. This completes the PC increment operation and

PC will now point to next instruction.

Step3--> Fetched instruction is moved into MDR and then to IR.

The step 1 through 3 constitutes the **Fetch Phase**.

At the beginning of step 4, the instruction decoder interprets the contents of the IR. This enables the control circuitry to activate the control-signals for steps 4 through 7.

The step 4 through 7 constitutes the **Execution Phase**.

Step4--> Contents of R3 are loaded into MAR & a memory read signal is issued. Step5--> Contents of R1 are transferred to Y to prepare for addition.

Step6--> When Read operation is completed, memory-operand is available in MDR, and the addition is performed.

Step7--> Sum is stored in Z, then transferred to R1.The End signal causes a new instruction

fetch cycle to begin by returning to step1.

BRANCHING INSTRUCTIONS

• Control sequence for an **unconditional branch instruction** is as follows:

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, Add, Z_{in}
5	Z_{out}, PC_{in}, End

Figure 3.7 Control sequence for an unconditional branch instructions.

• Instruction execution proceeds as follows:

Step 1-3--> The processing starts & the fetch phase ends in step3.

Step 4--> The offset-value is extracted from IR by instruction-decoding circuit.

Since the updated value of PC is already available in register Y, the offset X is gated onto the bus, and an addition operation is performed.

Step 5--> the result, which is the branch-address, is loaded into the PC.

• The branch instruction loads the branch target address in PC so that PC will fetch the next instruction from the branch target address.

• The branch target address is usually obtained by adding the offset in the contents of PC.

• The offset X is usually the difference between the branch target-address and the address immediately following the branch instruction.

• In case of **conditional branch**,

we have to check the status of the condition-codes before loading a new value into the PC. e.g.: Offset-field-of-IR_{out}, Add, Z_{in}, If N=0 then End

If N=0, processor returns to step 1 immediately after step 4.

If N=1, step 5 is performed to load a new value into PC.

MULTIPLE BUS ORGANIZATION

• **Disadvantage of Single-bus organization:** Only one data-word can be transferred over the bus in a clock cycle. This increases the steps required to complete the execution of the instruction

Solution: To reduce the number of steps, most processors provide multiple internal-paths. Multiple

paths enable several transfers to take place in parallel.

• As shown in fig 3.9, three buses can be used to connect registers and the ALU of the processor.

• All general-purpose registers are grouped into a single block called the **Register File**.

• Register-file has 3 ports:

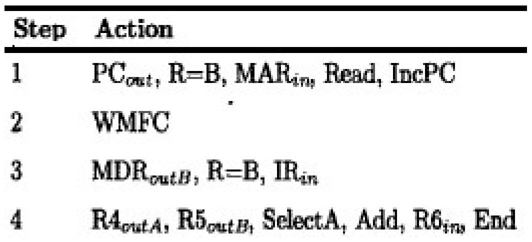
1) Two output-ports allow the contents of 2 different registers to be simultaneously placed on buses A & B.

2) Third input-port allows data on bus C to be loaded into a third register during the same clock-cycle.

• Buses A and B are used to transfer source-operands to A & B inputs of ALU.

• The result is transferred to destination over bus C.

• **Incrementer Unit** is used to increment PC by 4.



Step	Action
1	PC_{out}, R=B, MAR_{in}, Read, IncPC
2	WMFC
3	MDR_{outB}, R=B, IR_{in}
4	$R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End

Figure 3.8 control sequence of ADD R4,R5,R6

• Instruction execution proceeds as follows: Step 1--> Contents of PC are

→ passed through ALU using R=B control-signal &

→ loaded into MAR to start memory Read operation. At the same time, PC is incremented by 4. Step2--> Processor waits for MFC signal from memory.

Step3--> Processor loads requested-data into MDR, and then transfers them to IR.

Step4--> The instruction is decoded and add operation takes place in a single step.

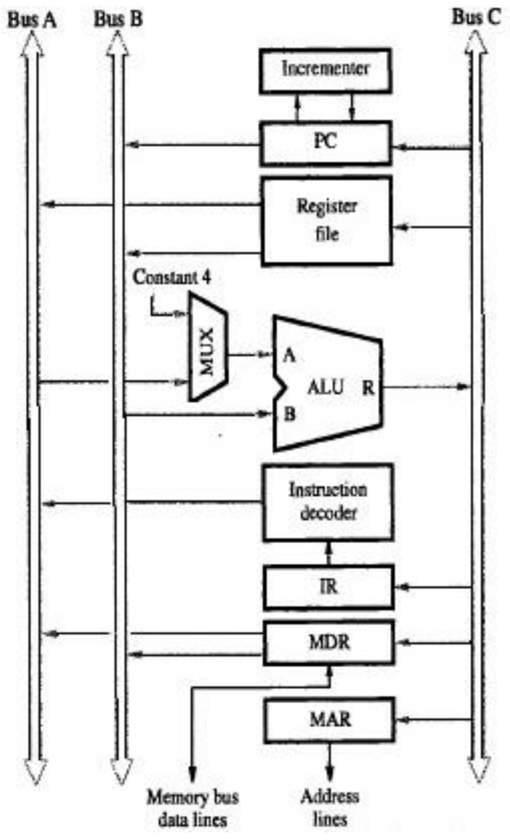


Figure 3.9 Three bus organization of the data path

COMPLETE PROCESSOR

• This has separate processing-units to deal with integer data and floating-point data.

Integer Unit To process integer data. (Figure 3.10).

Floating Unit To process floating –point data.

• **Data-Cache** is inserted between these processing-units & main-memory.

The integer and floating unit gets data from data cache.

• **Instruction-Unit** fetches instructions

→ from an instruction-cache or

→ from main-memory when desired instructions are not already in cache.

• Processor is connected to system-bus &

hence to the rest of the computer by means of a **Bus Interface.**

• Using separate caches for instructions & data is common practice in many processors today.

• A processor may include several units of each type to increase the potential for concurrent operations.

• The 80486 processor has 8-kbytes single cache for both instruction and data.

Whereas the Pentium processor has two separate 8 kbytes caches for instruction and data

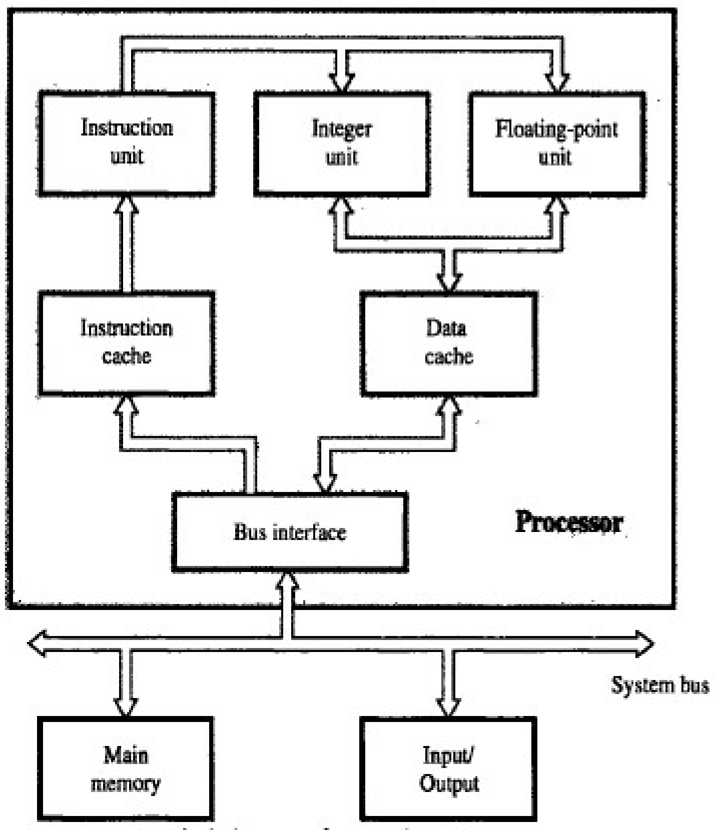


Figure 3.10Block diagram of complete processor

Note:

To execute instructions, the processor must have some means of generating the control-signals. There are two approaches for this purpose:

1) Hardwired control and 2) Microprogrammed control.

HARDWIRED CONTROL

HARDWIRED CONTROL

• Hardwired control is a method of control unit design (Figure 3.11).

• The control-signals are generated by using logic circuits such as gates, flip-flops, decoders etc.

• **Decoder/Encoder Block** is a combinational-circuit that generates required control-outputs depending on state of all its inputs.

• **Instruction Decoder**

It decodes the instruction loaded in the IR.

If IR is an 8 bit register, then instruction decoder generates 2^8(256 lines); one for each instruction.

It consists of a separate output-lines INS_1 through INS_m for each machine instruction.

According to code in the IR, one of the output-lines INS_1 through INS_m is set to 1, and all

other lines are set to 0.

• **Step-Decoder** provides a separate signal line for each step in the control sequence.

• **Encoder**

It gets the input from instruction decoder, step decoder, external inputs and condition codes.

It uses all these inputs to generate individual control-signals: Y_{in}, PC_{out}, Add, End and so on.

For example (Figure 3.12), $Z_{in}=T_1+T_6.ADD+T_4.BR$

;This signal is asserted during time-slot T_1 for all instructions.

during T_6 for an Add instruction.

during T_4 for unconditional branch instruction

• When **RUN**=1, counter is incremented by 1 at the end of every clock cycle.

When RUN=0, counter stops counting.

• After execution of each instruction, **end** signal is generated. End signal resets step counter.

• Sequence of operations carried out by this machine is determined by wiring of logic circuits, hence

the name “**hardwired**”.

• **Advantage**: Can operate at high speed.

• **Disadvantages:**

1) Since no. of instructions/control-lines is often in hundreds, the complexity of control unit is very high.
2) It is costly and difficult to design.
3) The control unit is inflexible because it is difficult to change the design.

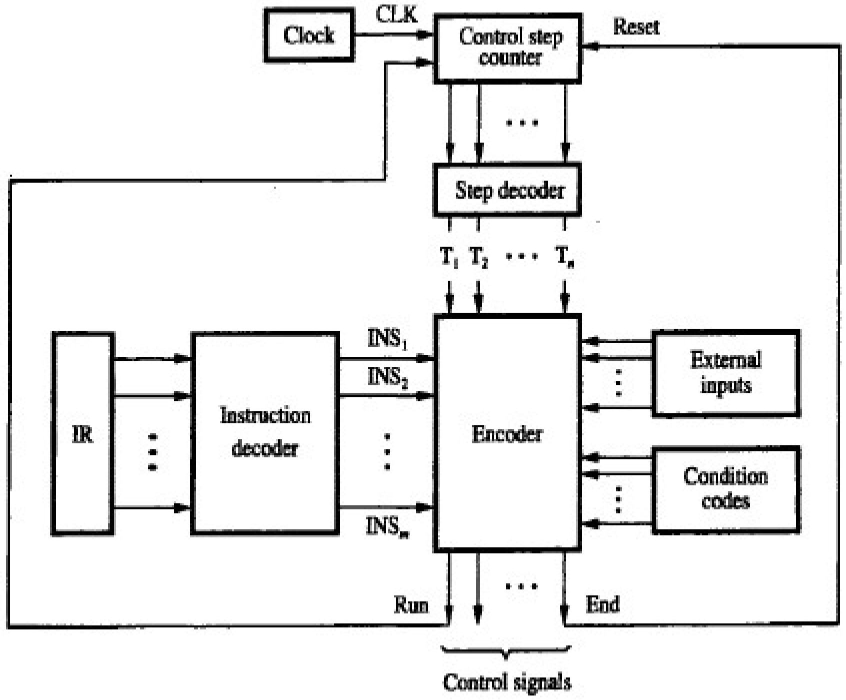


Figure 3.11 Separation of the decoding and encoding function

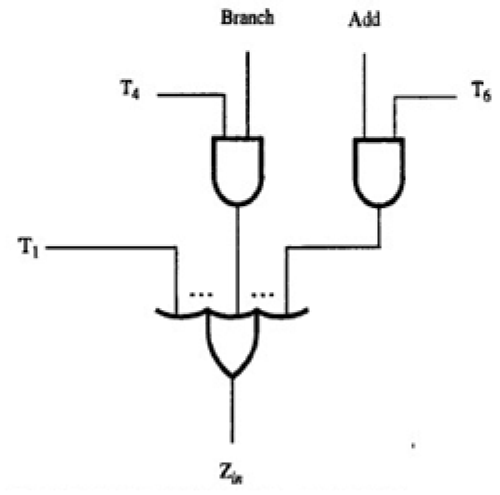


Figure 3.12 generation of Z_{in} control signal

HARDWIRED CONTROL VS MICROPROGRAMMED CONTROL

Attribute	**Hardwired Control**	**Microprogrammed Control**
Definition	Hardwired control is a control mechanism to generate control- signals by using	Micro programmed control is a control mechanism to generate control-signals by using a memory called
Speed	Fast	Slow
Control functions	Implemented in hardware.	Implemented in software.
Flexibility	Not flexible to accommodate	More flexible, to accommodate new
Ability to handle large	Difficult.	Easier.
Ability to support	Very difficult.	Easy.
Design process	Complicated.	Orderly and systematic.
Applications	Mostly RISC microprocessors.	Mainframes, some microprocessors.
Instructionset size	Usually under 100	Usually over 100 instructions.
ROM size	**-**	2K to 10K by 20-400 bit microinstructions.
Chip area efficiency	Uses least area.	Uses more area.

MICROPROGRAMMED CONTROL

• Microprogramming is a method of control unit design (Figure 7.16).

• Control-signals are generated by a program similar to machine language programs.

• **Control Word(CW*)*** is a word whose individual bits represent various control-signals (like Add, PC_{in}).

• Each of the control-steps in control sequence of an instruction defines a unique combination of 1s &

0s in CW.

• Individual control-words in microroutine are referred to as **microinstructions** (Figure 3.14).

• A sequence of CWs corresponding to control-sequence of a machine instruction constitutes the

microroutine.

• The microroutines for all instructions in the instruction-set of a computer are stored in a special memory called the **Control Store (CS)***.*

• Control-unit generates control-signals for any instruction by sequentially reading CWs of

corresponding microroutine from CS.

• **µPC** is used to read CWs sequentially from CS. (µPC Microprogram Counter).

• Every time new instruction is loaded into IR, o/p of **Starting Address Generator** is loaded into µPC.

• Then, µPC is automatically incremented by clock; causing successive microinstructions to be read from CS. Hence, control-signals are delivered to various parts of processor in correct sequence.

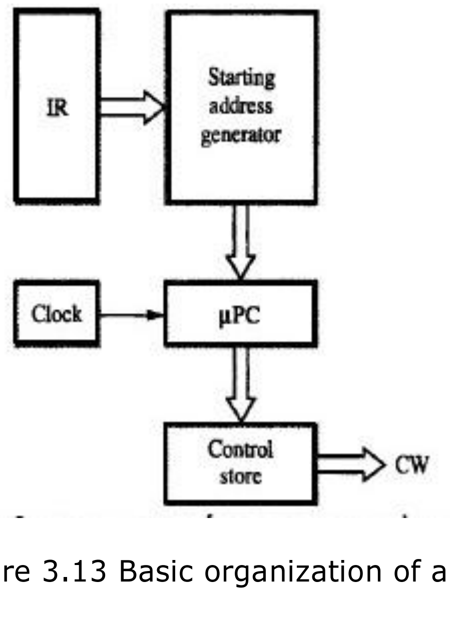


Figure 3.13 Basic organization of a microprogrammed control unit

Micro - instruction	..	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	WMFC	End	..
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	

Figure 3.14 An example of micro instructions

Advantages

• It simplifies the design of control unit. Thus it is both, cheaper and less error prone implement.

• Control functions are implemented in software rather than hardware.

• The design process is orderly and systematic.

• More flexible, can be changed to accommodate new system specifications or to correct the design errors quickly and cheaply.

• Complex function such as floating point arithmetic can be realized efficiently.

Disadvantages

• A microprogrammed control unit is somewhat slower than the hardwired control unit, because time is

required to access the microinstructions from CM.

• The flexibility is achieved at some extra hardware cost due to the control memory and its access circuitry.

ORGANIZATION OF MICROPROGRAMMED CONTROL UNIT TO SUPPORT CONDITIONAL BRANCHING

• **Drawback of previous Microprogram control:**

It cannot handle the situation when the control unit is required to check the status of the condition codes or external inputs to choose between alternative courses of action.

Solution:

Use conditional branch microinstruction.

• In case of conditional branching, microinstructions specify which of the external inputs, condition- codes should be checked as a condition for branching to take place.

• **Starting and Branch Address Generator Block** loads a new address into μPC when a microinstruction instructs it to do so (Figure 7.18).

• To allow implementation of a conditional branch, inputs to this block consist of

→ external inputs and condition-codes &

→ contents of IR.

• μPC is incremented every time a new microinstruction is fetched from microprogram memory except in following situations:

1) When a new instruction is loaded into IR, μPC is loaded with starting-address of microroutine for that instruction.

2) When a Branch microinstruction is encountered and branch condition is satisfied, μPC is loaded with branch-address.

3) When an End microinstruction is encountered, μPC is loaded with address of first CW in microroutine for instruction fetch cycle.

Address	Microinstruction
0	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
1	Z_{out}, PC_{in}, Y_{in}, WMFC
2	MDR_{out}, IR_{in}
3	Branch to starting address of appropriate microroutine
25	If N=0, then branch to microinstruction 0
26	Offset-field-of-IR_{out}, SelectY, Add, Z_{in}
27	Z_{out}, PC_{in}, End

Figure 3.15 Microroutine for the instruction branch <0

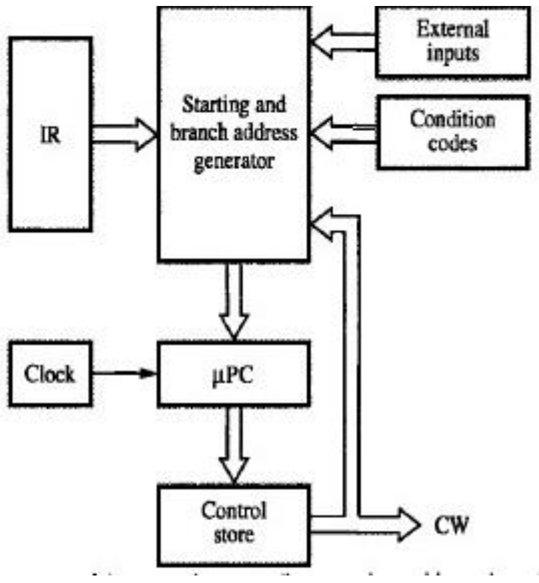


Figure 3.16 organization of the control unit to allow conditional branching

MICROINSTRUCTIONS

• A simple way to structure microinstructions is to assign one bit position to each control-signal required in the CPU.

• There are 42 signals and hence each microinstruction will have 42 bits.

• **Drawbacks of microprogrammed control:**

1) Assigning individual bits to each control-signal results in long microinstructions because the number of required signals is usually large.

2) Available bit-space is poorly used because

only a few bits are set to 1 in any given microinstruction.

• **Solution:** Signals can be grouped because

1) Most signals are not needed simultaneously.

2) Many signals are mutually exclusive. E.g. only 1 function of ALU can be activated at a time.

For ex: Gating signals: IN and OUT signals (Figure 7.19).

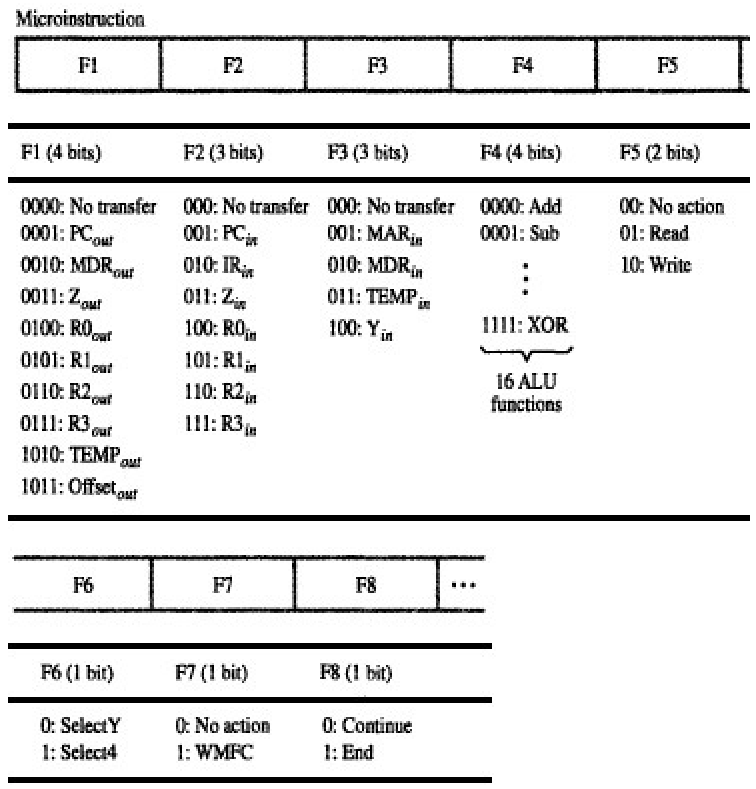
Control-signals: Read, Write.

ALU signals: Add, Sub, Mul, Div, Mod.

• Grouping control-signals into fields requires a little more hardware because

decoding-circuits must be used to decode bit patterns of each field into individual control-signals.

• **Advantage:** This method results in a smaller control-store (only 20 bits are needed to store the patterns for the 42 signals).



Microinstruction

F1	F2	F3	F4	F5

F1 (4 bits)	F2 (3 bits)	F3 (3 bits)	F4 (4 bits)	F5 (2 bits)
0000: No transfer	000: No transfer	000: No transfer	0000: Add	00: No action
0001: PC_{out}	001: PC_{in}	001: MAR_{in}	0001: Sub	01: Read
0010: MDR_{out}	010: IR_{in}	010: MDR_{in}	⋮	10: Write
0011: Z_{out}	011: Z_{in}	011: $TEMP_{in}$		
0100: $R0_{out}$	100: $R0_{in}$	100: Y_{in}	1111: XOR	
0101: $R1_{out}$	101: $R1_{in}$		16 ALU functions	
0110: $R2_{out}$	110: $R2_{in}$			
0111: $R3_{out}$	111: $R3_{in}$			
1010: $TEMP_{out}$				
1011: $Offset_{out}$				

F6	F7	F8	•••

F6 (1 bit)	F7 (1 bit)	F8 (1 bit)
0: SelectY	0: No action	0: Continue
1: Select4	1: WMFC	1: End

Figure 3.17an example of partial format for field encoded microinstructions

TECHNIQUES OF GROUPING OF CONTROL-SIGNALS

• The grouping of control-signal can be done either by using

1) Vertical organization &
2) Horizontal organization.

Vertical Organization	Horizontal Organization
Highly encoded schemes that use compact codes to specify only a small number of control	The minimally encoded scheme in which many resources can be controlled with a single
Slower operating-speeds.	Useful when higher operating-speed is
Short formats.	Long formats.
Limited ability to express parallel microoperations.	Ability to express a high degree of parallelism.
Considerable encoding of the control information.	Little encoding of the control information.

MICROPROGRAM SEQUENCING

• The task of microprogram sequencing is done by microprogram sequencer.

• Two important factors must be considered while designing the microprogram sequencer:

1) The size of the microinstruction &

2) The address generation time.

• The size of the microinstruction should be minimum so that the size of control memory required to store microinstructions is also less.

• This reduces the cost of control memory.

• With less address generation time, microinstruction can be executed in less time resulting better throughout.

• During execution of a microprogram the address of the next microinstruction to be executed has 3 sources:

1) Determined by instruction register.

2) Next sequential address &

3) Branch.

• Microinstructions can be shared using microinstruction branching.

• **Disadvantage of microprogrammed branching**:

1) Having a separate microroutine for each machine instruction results in a large total number of microinstructions and a large control-store.

2) Execution time is longer because it takes more time to carry out the required branches.

• Consider the instruction *Add src,Rdst* ;which adds the source-operand to the contents of Rdst and places the sum in Rdst.

• Let source-operand can be specified in following addressing modes (Figure 3.18):

a) Indexed
b) Autoincrement
c) Autodecrement

d) Register indirect &

e) Register direct

• Each box in the chart corresponds to a microinstruction that controls the transfers and operations indicated within the box.

• The microinstruction is located at the address indicated by the octal number (001,002).

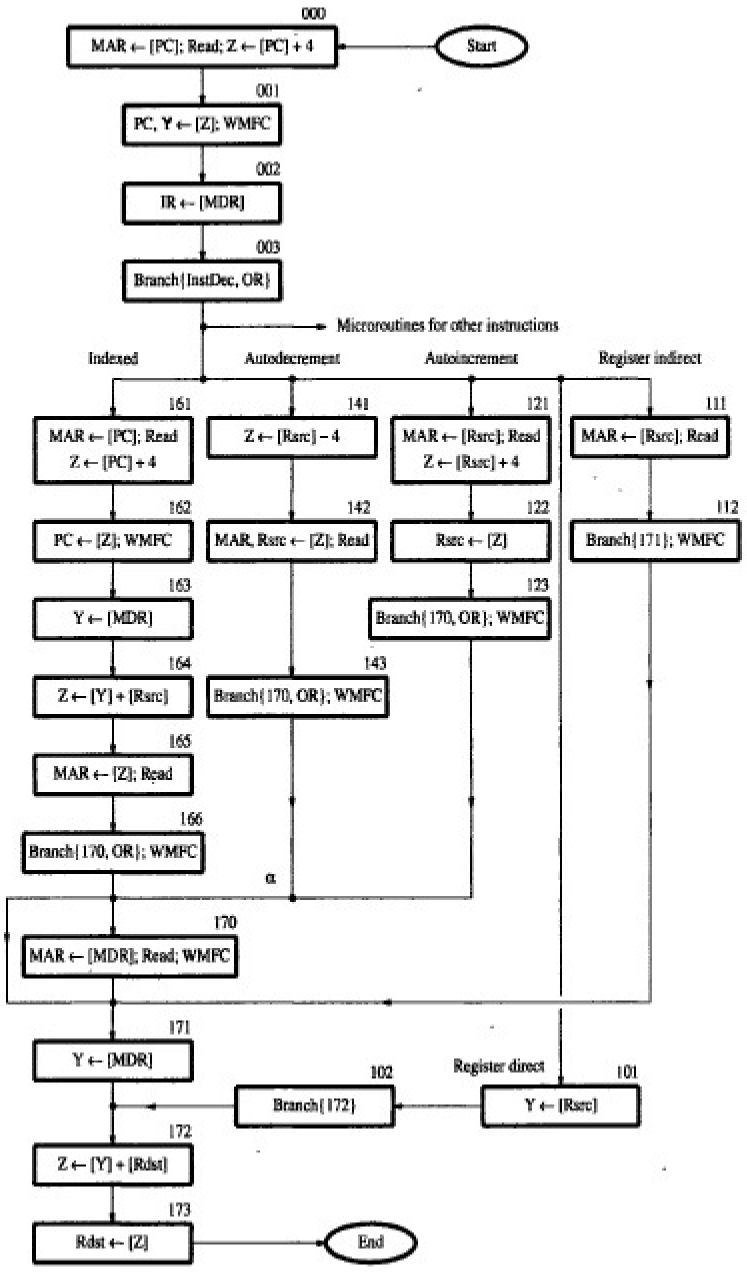


Figure 3.18 Flowchart of a micro program for the ADD SRC, Rdst instruction.

BRANCH ADDRESS MODIFICATION USING BIT-ORING

• The branch address is determined by ORing particular bit or bits with the current address of microinstruction.

• **Eg:** If the current address is 170 and branch address is 171 then the branch address can be

generated by ORing 01(bit 1), with the current address.

• Consider the point labeled α in the figure. At this point, it is necessary to choose between direct and indirect addressing modes.

• If indirect-mode is specified in the instruction, then the microinstruction in location 170 is performed

to fetch the operand from the memory.

If direct-mode is specified, this fetch must be bypassed by branching immediately to location 171.

• The most efficient way to bypass microinstruction 170 is

to have bit-ORing of

→ current address 170 &

→ branch address 171.

WIDE BRANCH ADDRESSING

• The instruction-decoder (InstDec) generates the starting-address of the microroutine that implements the instruction that has just been loaded into the IR.

• Here, register IR contains the Add instruction, for which the instruction decoder generates the

microinstruction address 101. (However, this address cannot be loaded as is into the μPC).

• The source-operand can be specified in any of several addressing-modes. The bit-ORing technique can be used to modify the starting-address generated by the instruction-decoder to reach the appropriate path.

USE of WMFC:

• WMFC signal is issued at location 112 which causes a branch to the microinstruction in location 171.

• WMFC signal means that the microinstruction may take several clock cycles to complete. If the branch is allowed to happen in the first clock cycle, the microinstruction at location 171 would be

fetched and executed prematurely. To avoid this problem, WMFC signal must inhibit any change in the

contents of the μPC during the waiting-period.

Detailed Examination of Add (Rsrc)+,Rdst

• Consider *Add (Rsrc)+,Rdst;* which adds Rsrc content to Rdst content, then stores the sum in Rdst and finally increments Rsrc by 4 (i.e. auto-increment mode).

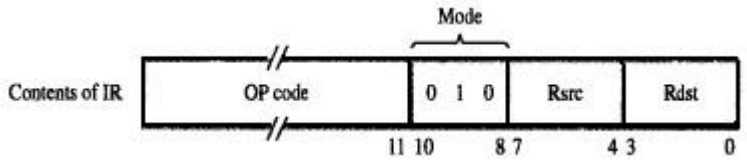
• In bit 10 and 9, bit-patterns 11, 10, 01 and 00 denote indexed, auto-decrement, auto-increment and

register modes respectively. For each of these modes, bit 8 is used to specify the indirect version.

• The processor has 16 registers that can be used for addressing purposes; each specified using a 4- bit-code (Figure 3.19).

• There are 2 stages of decoding:

1) The microinstruction field must be decoded to determine that an Rsrc or Rdst register is involved.
2) 2) The decoded output is then used to gate the contents of the Rsrc or Rdst fields in the IR into a second decoder, which produces the gating-signals for the actual registers R0 to R15.


Mode
Contents of IR
OP code
0 1 0
Rsrc
Rdst
11 10
8 7
4 3
0