COMPUTER ENGINEERING DEPARTMENT

ASSIGNMENT NO-02

SUB: Microprocessor

COURSE: T.E. **Year: 2020-2021** **Semester: V**

DEPT: Computer Engineering

SUBJECT CODE: CSC501 **SUBMISSION DATE: 26/11/2020**

===

Name: Amey Thakur **Roll No.:** 50

Batch: B3 **Class:** TE COMPS B

Assignment II

Q. No	Questions	CO Mapping	PO Mapping
1.	List out various DMAC Operating modes.	4	1,2,3,4,5,8, 10
2.	Design 8086microprocessor system having 64 KB RAM and 64 KB ROM assume frequency 5MHz.	4	1,2,3,4,5,8, 10
3.	Explain Architecture of 80386 DX microprocessor.	5	1,2,3,4,5,10
4.	Explain Real mode, protected mode and virtual 8086 modes of 80386 DX processor.	5	1,2,3,4,5,10
5.	Describe the Programming model of 80386DX processor.	6	1,2,3,4,5,10
6.	How the flushing of the pipeline problem is minimized in Pentium architecture?	6	1,2,3,4,5,10

1. **List out various DMAC Operating modes.**

Ans:

Operation Cycles of DMAC

There are mainly two operation cycles of a DMAC.

1. **Idle Cycle**
   After Reset, the DMAC is in idle state (idle cycle).
   During idle state, no DMA operation is taking place.
   No DMA requests are active.
   The initialization of the DMAC takes place in idle mode.

2. **Active Cycle**
   Once DMA operation begins, the DMAC is said to be in active mode.
   Now the DMAC controls the system bus.
   There are three types of ACTIVE DMA Cycles while performing DMA transfer:

1. **DMA Read**
   The DMAC reads data from the memory and writes into the I/O device.
   Thus, MEMR and IOW signals are used.

2. **DMA Write**
   The DMAC reads data from the I/O device and writes into the memory.
   Thus, IOR and MEMW signals are used.

3. **DMA Verify**
   In this cycle, 8237 does not generate any control signals.
   Hence, no data transfer takes place.
   During this time, the peripheral and the DMAC verify the correctness of the data transferred, using some error detection method.

2. **Design 8086 microprocessor system having 64 KB RAM and 64 KB ROM assume frequency 5MHz.**

Ans:

Memory Calculations:

EPORM:

Required = 64 KB, Available = 16 KB
No. of chips = 4 chips.

Starting address of EPROM is calculated as:
FFFFFH – (Space required by total EPROM of 64 KB)

```
F F F F F H
- F F F F H
----------------
F 0 0 0 0 H
```

Size of a single EPROM chip = 16 KB
= 16 x 1KB
= 2^4 x 2^{10}
= 2^{14}
= 14 address lines
= (A 14 … A1)

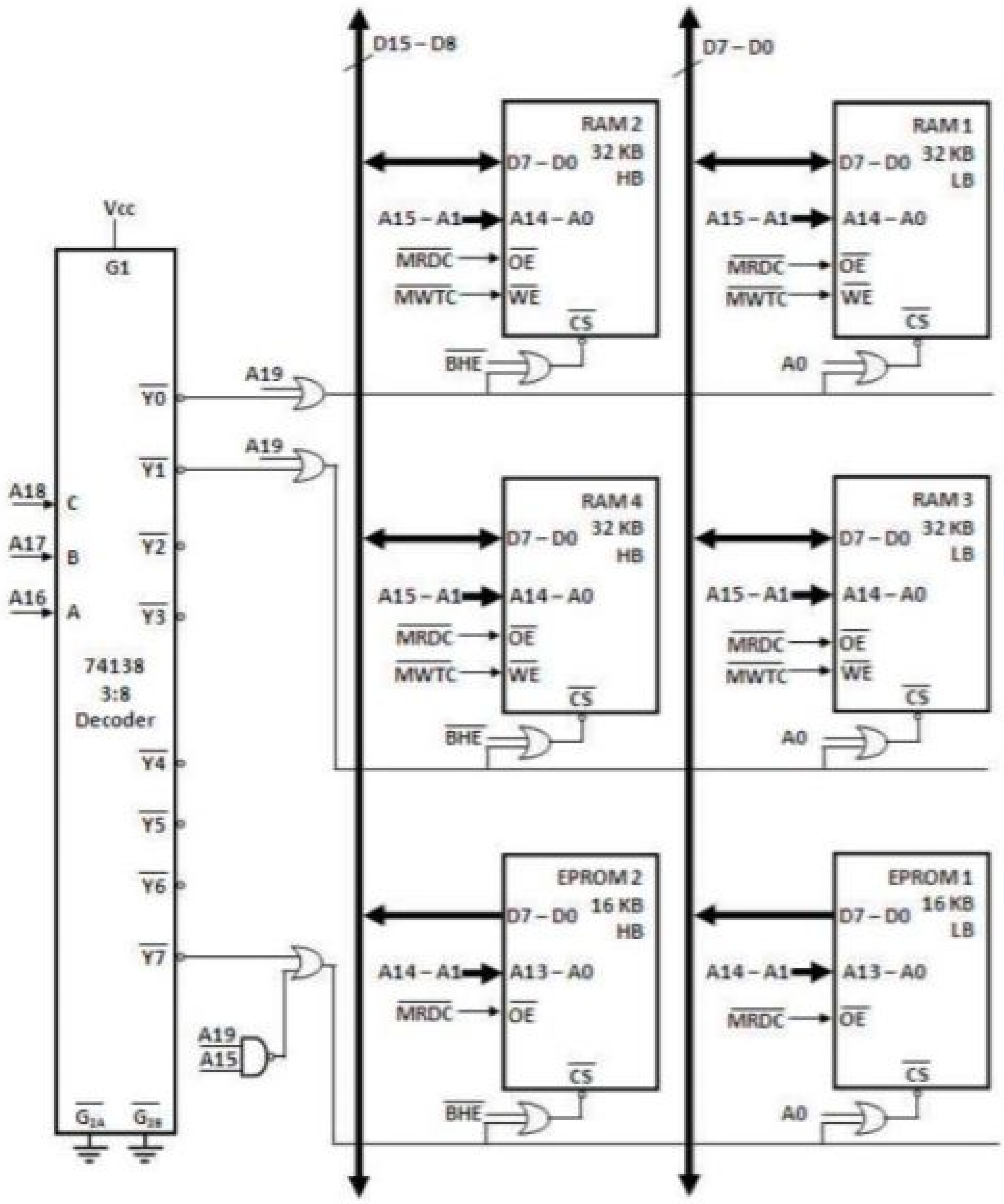
RAM:

Required = 64 KB, Available = 16 KB
No. of chips = 4 chips.

Starting address of RAM is: 00000H
Size of a single RAM chip = 16 KB
= 16 x 1 KB
= 2^4 x 2^{10}
= 2^{14}
= 15 address lines
= (A14 … A1)

Memory Map

Memory	Address Bus																				Memory
Chip	A19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	A0	Address
RAM 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00000H
(LB)	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0FFFEH
RAM 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	00001H
(HB)	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0FFFFH
RAM 3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10000H
(LB)	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1FFFEH
RAM 4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	10001H
(HB)	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFFH
EPORM 1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	F8000H
(LB)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	FFFFEH
EPORM 2	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	F8001H
(HB)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFFFH

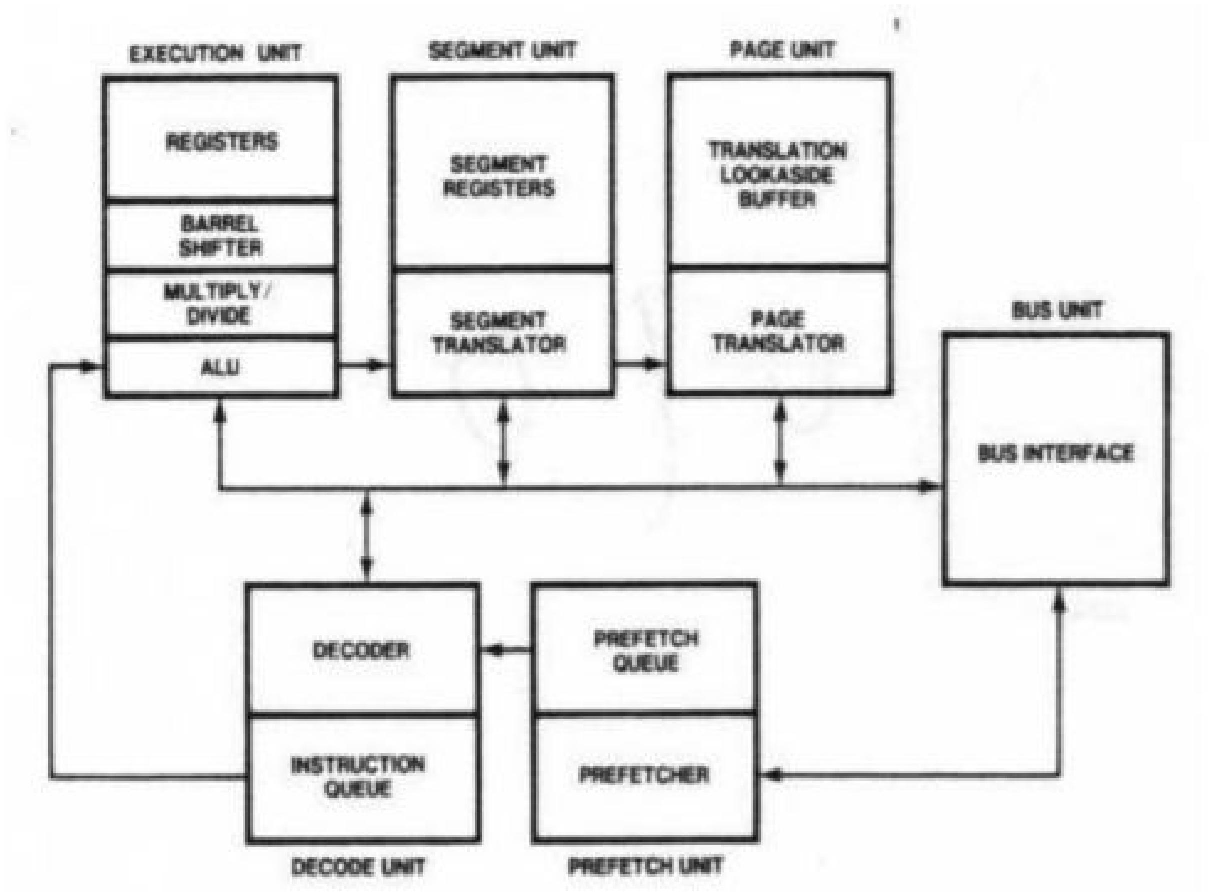


D15 – D8
D7 – D0
RAM 2 32 KB HB
RAM 1 32 KB LB
RAM 4 32 KB HB
RAM 3 32 KB LB
EPROM 2 16 KB HB
EPROM 1 16 KB LB
D7 – D0
A15 – A1
A14 – A0
A14 – A1
A13 – A0
MRDC
MWTC
OE
WE
CS
BHE
A0
A19
A15
Vcc
G1
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
A18 C
A17 B
A16 A
74138 3:8 Decoder
G2A
G2B

3. **Explain Architecture of 80386 DX microprocessor.**

Ans:

80386 Architecture



1. **Bus Unit (Bus Interface Unit)**

➔ The Bus unit is responsible for transferring data in and out of the µP.
➔ It is connected to the external memory and I/O devices, using the system bus.
➔ It gets requests from the Prefetch unit for fetching instructions and from the execution unit for transferring data.
➔ If both requests occur simultaneously preference is given to the execution unit.

2. **Prefetch Unit**

➔ The Prefetch unit fetches further instructions in advance to implement pipelining.
➔ It fetches the next 16 bytes of the program and stores it into the Prefetch Queue.
➔ It refills the queue when at least 4 bytes are empty as 80386 has a 32-bit data bus.
➔ During a branch, the instructions in the queue are invalid and hence are discarded.

3. **Decode Unit**

➔ 80386 μP has a separate unit for decoding instructions called the Decode Unit.
➔ It decodes the next three instructions and keeps them ready in the Decode Queue.
➔ The decoded instructions are stored in Micro-Coded form.
➔ During a branch, the instructions in the queue are invalid and hence are discarded.

4. **Execution Unit**

➔ The Execution Unit performs the main task of executing instructions.
➔ Normally, execution requires Arithmetic or Logic operations performed by a 32-bit ALU.
➔ It also has dedicated circuits for 32-bit multiplication and division. 4) A 64-bit barrel shifter is also provided for faster shifts during multiplication and division.
➔ Operands For the ALU can either be provided in the instruction, or can be taken from memory or could be taken from the 32-bit registers like EAX, EBX etc.
➔ Additionally, there is a 32-bit Flag register (EFLAGS) giving the Status of the current result.

5. **Memory Unit**

➔ The Memory unit converts Virtual Address (Logical address) to Physical Address.
➔ 80386 μP implements 64 Terabytes of Virtual memory using Segmentation and Paging. Hence the Memory Unit is subdivided into Segmentation Unit and Paging Unit.
➔ Segmentation is compulsory, while Paging is optional.
➔ The Segmentation Unit converts the Logical Address into a Linear Address.
➔ The Paging Unit converts the Linear Address into a Physical Address.
➔ If Paging is not used, then the Linear Address itself is the Physical Address

4. **Explain Real mode, protected mode and virtual 8086 modes of 80386 DX processor.**

Ans:

➔ In the protected mode, 80386 microprocessor operates in a similar way like 80286, but offers higher memory addressing ability.In protected mode of operation, 80386DX provides a virtual 8086 operating environment to execute the 8086 programs.

➔ The real mode can also be used to execute the 8086 programs along with the capabilities of 80386, like protection and a few additional instructions.Once the 80386 enters the protected mode from the real mode, it cannot return back to the real mode without a reset operation.

➔ In virtual mode, the overall memory of 80386 can be divided into various virtual machines. And all of them act as a separate computer with 8086 microprocessors. This mode is also called virtual 8086 mode or V86 mode.

➔ The other one is the virtual real mode, this mode allows the system to execute multiple programs in the protected memory. And in case a program at a particular memory gets crashed then it will not cause any adverse effect on the other part of the memory.the virtual 8086 mode of operation of 80386, offers an advantage of executing 8086 programs while in protected mode.V86 Mode is also known as Virtual Mode of 80386.

➔ V86 Mode is a Dynamic Mode.It can switch repeatedly & rapidly between V86 Mode & Protected Mode.To execute an 8086 program, the CPU enters in V86 Mode from Protected Mode.CPU Leaves V86 Mode and enters protected mode to continue executing a native 80386 program.The address forming mechanism in virtual 8086 mode is exactly identical with that of 8086 real mode.In virtual mode, 8086 can address 1MB of physical memory that may be anywhere in the 4GB address space of the protected mode of 80386.Like 80386 real mode, the addresses in virtual 8086 mode lie within 1MB of memory.In virtual mode, the paging mechanism and protection capabilities are available at the service of the programmers.The virtual mode allows the multiprogramming of 8086 applications.The virtual 8086 mode executes all the programs at privilege level 3.

5. **Describe the Programming model of 80386DX processor.**

Ans:

The basic programming model consists of these aspects:

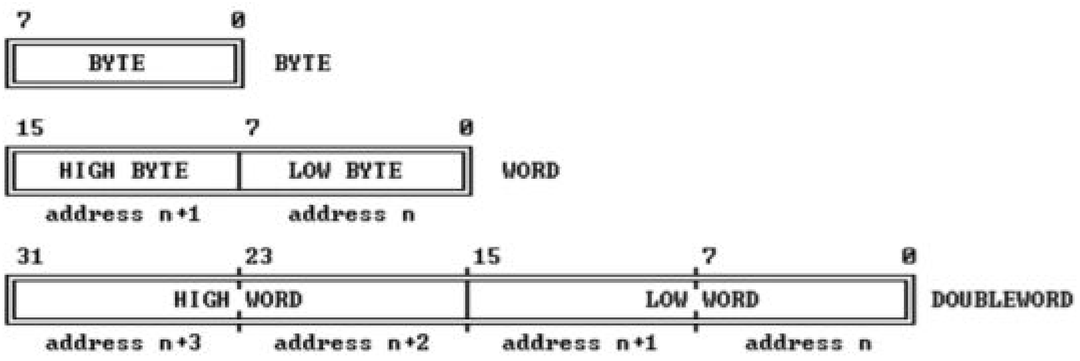
1. Memory organization and segmentation
2. Data types
3. Registers
4. Instruction format
5. Operand selection
6. Interrupts and exceptions
7. Memory organization and segmentation

Memory organization and segmentation

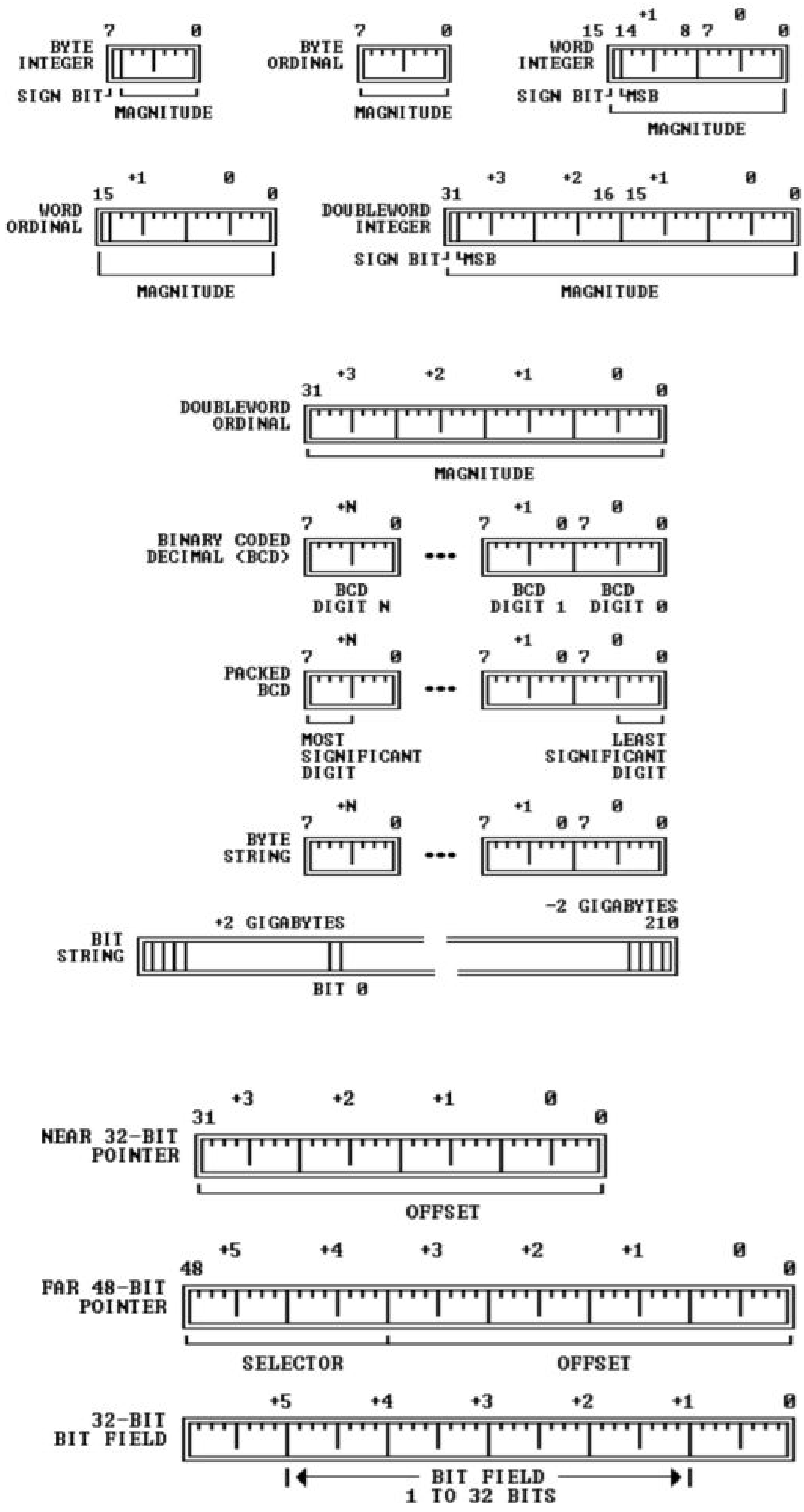
- ➔ The physical memory of an 80386 system is organized as a sequence of 8-bit bytes. Each byte is assigned a unique address that ranges from zero to a maximum of 2^(32) -1 (4 gigabytes).
- ➔ 80386 programs, however, are independent of the physical address space. This means that programs can be written without knowledge of how much physical memory is available and without knowledge of exactly where in physical memory the instructions and data are located.
- ➔ The model of memory organization seen by applications programmers is determined by systems-software designers. The architecture of the 80386 gives designers the freedom to choose a model for each task. The model of memory organization can range between the following extremes:
  - ◆ A "flat" address space consisting of a single array of up to 4 gigabytes.
  - ◆ A segmented address space consisting of a collection of up to 16,383 linear address spaces of up to 4 gigabytes each.
- ➔ Both models can provide memory protection. Different tasks may employ different models of memory organization.

Data types

- ➔ Fundamental data types



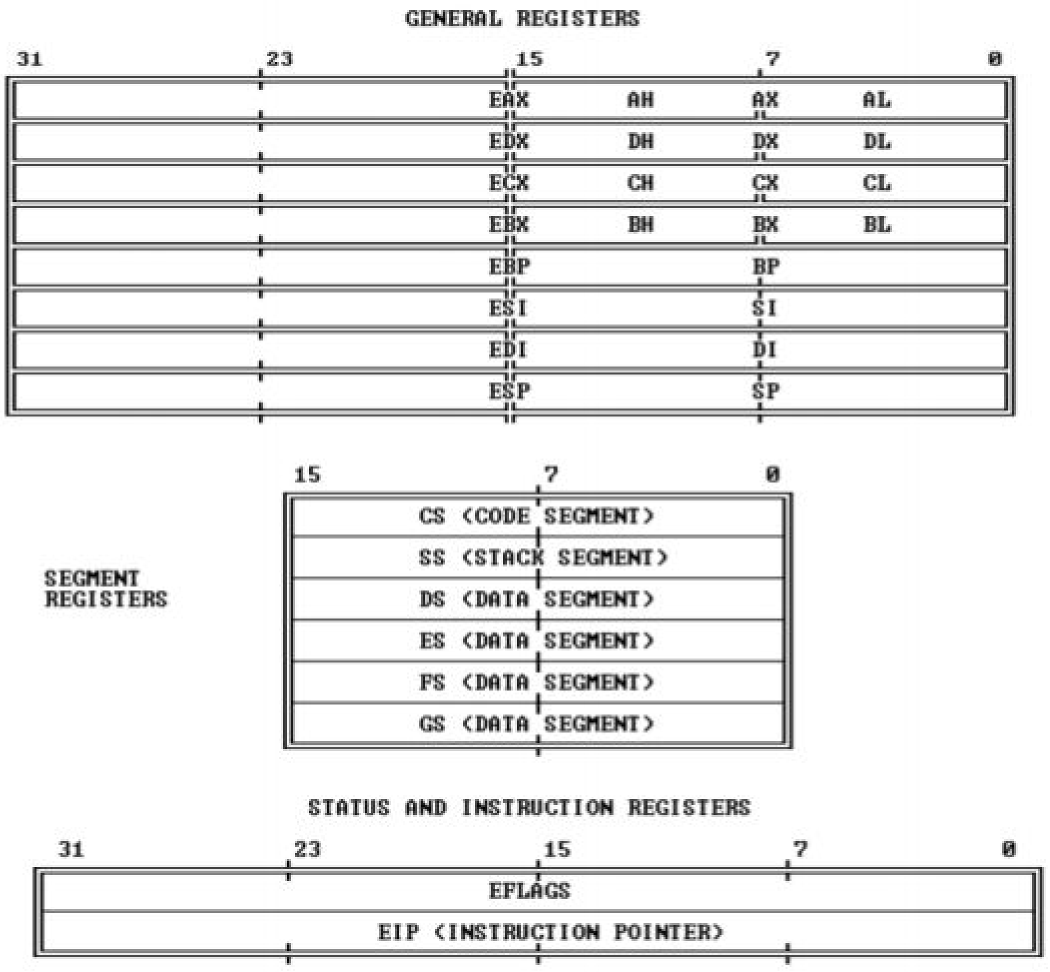
→ 80386 Data Types



Registers

The 80386 contains a total of sixteen registers that are of interest to the applications programmer, these registers may be grouped into these basic categories:

- ➔ General registers. These eight 32-bit general-purpose registers are used primarily to contain operands for arithmetic and logical operations.
- ➔ Segment registers. These special-purpose registers permit systems software designers to choose either a flat or segmented model of memory organization. These six registers determine, at any given time, which segments of memory are currently addressable.
- ➔ Status and instruction registers. These special-purpose registers are used to record and alter certain aspects of the 80386 processor state.



Instruction Format

The information encoded in an 80386 instruction includes a specification of the operation to be performed, the type of the operands to be manipulated, and the location of these operands. 80386 instructions are composed of various elements and have various formats; the elements of instructions are described below. Of these instruction elements, only one, the opcode, is always present. The other elements may or may not be present, depending on the particular operation involved and, on the location, and type of the operands. The elements of an instruction, in order of occurrence are as follows:

- ➔ Prefixes -- one or more bytes preceding an instruction that modify the operation of the instruction. The following types of prefixes can be used by applications programs:

    1. Segment override -- explicitly specifies which segment register an instruction should use, thereby overriding the default segment-register selection used by the 80386 for that instruction.

    2. Address size -- switches between 32-bit and 16-bit address generation.

    3. Operand size -- switches between 32-bit and 16-bit operands.

    4. Repeat -- used with a string instruction to cause the instruction to act on each element of the string.

- ➔ Opcode -- specifies the operation performed by the instruction. Some operations have several different opcodes, each specifying a different variant of the operation.
- ➔ Register specifier -- an instruction may specify one or two register operands. Register specifiers may occur either in the same byte as the opcode or in the same byte as the addressing-mode specifier.
- ➔ Addressing-mode specifier -- when present, specifies whether an operand is a register or memory location; if in memory, specifies whether a displacement, a base register, an index register, and scaling are to be used.
- ➔ SIB (scale, index, base) byte -- when the addressing-mode specifier indicates that an index register will be used to compute the address of an operand, an SIB byte is included in the instruction to encode the base register, the index register, and a scaling factor.
- ➔ Displacement -- when the addressing-mode specifier indicates that a displacement will be used to compute the address of an operand, the displacement is encoded in the instruction. A displacement is a signed integer of 32, 16, or eight bits. The eightbit form is used in the common case when the displacement is sufficiently small. The processor extends an eight-bit displacement to 16 or 32 bits, taking into account the sign.
- ➔ Immediate operand -- when present, directly provides the value of an operand of the instruction. Immediate operands may be 8, 16, or 32 bits wide. In cases where an eight-bit immediate operand is combined in some way with a 16- or 32-bit operand, the processor automatically extends the size of the eight-bit operand, taking into account the sign

Operand Selection

An instruction can act on zero or more operands, which are the data manipulated by the instruction. An example of a zero-operand instruction is NOP (no operation). An operand can be in any of these locations:

- ➔ In the instruction itself (an immediate operand)
- ➔ In a register (EAX, EBX, ECX, EDX, ESI, EDI, ESP, or EBP in the case of 32-bit operands; AX, BX, CX, DX, SI, DI, SP, or BP in the case of 16-bit operands; AH, AL, BH, BL, CH, CL, DH, or DL in the case of 8-bit operands; the segment registers; or the EFLAGS register for flag operations)
- ➔ In memory
- ➔ At an I/O port

Immediate operands and operands in registers can be accessed more rapidly than operands in memory since memory operands must be fetched from memory. Register operands are available in the CPU. Immediate operands are also available in the CPU, because they are prefetched as part of the instruction.

Of the instructions that have operands, some specify operands implicitly; others specify operands explicitly; still others use a combination of implicit and explicit specification

Interrupts and exceptions

The 80386 has two mechanisms for interrupting program execution:

1. Exceptions are synchronous events that are the responses of the CPU to certain conditions detected during the execution of an instruction.

2. Interrupts are asynchronous events typically triggered by external devices needing attention.

Interrupts and exceptions are alike in that both cause the processor to temporarily suspend its present program execution in order to execute a program of higher priority. The major distinction between these two kinds of interrupts is their origin. An exception is always reproducible by re-executing with the program and data that caused the exception, whereas an interrupt is generally independent of the currently executing program.

6. **How the flushing of the pipeline problem is minimized in Pentium architecture?**

Ans:

- ➔ Performance gain through pipelining can be reduced by the presence of program transfer instructions (such as JMP, CALL, RET and conditional jumps).
- ➔ They change the sequence causing all the instructions that entered the pipeline after program transfer instruction invalid.
- ➔ Suppose instruction I3 is a conditional jump to I50 at some other address (target address), then the instructions that entered after I3 is an invalid and new sequence beginning with I50 need to be loaded in.
- ➔ This causes bubbles in the pipeline, where no work is done as the pipeline stages are reloaded.
- ➔ To avoid this problem, the Pentium uses a scheme called Dynamic Branch Prediction.
- ➔ In this scheme, a prediction is made concerning the branch instruction currently in the pipeline.
- ➔ The prediction will be either taken or not taken.
- ➔ If the prediction turns out to be true, the pipeline will not be flushed and no clock cycles will be lost. If the prediction turns out to be false, the pipeline is flushed and started over with the correct instruction.
- ➔ It results in a 3 cycle penalty if the branch is executed in the u-pipeline and 4 cycle penalty in v-pipeline.
- ➔ It is implemented using a 4-way set associative cache with 256 entries. This is referred to as the Branch Target Buffer (BTB).
- ➔ The directory entry for each line contains the following information:
- ➔ Valid Bit: Indicates whether or not the entry is in use.
- ➔ History Bits: track how often the branch has been taken.
- ➔ Source memory address that the branch instruction was fetched from (address of I3).
- ➔ If its directory entry is valid, the target address of the branch is stored in corresponding data entry in BTB.