LESSON 24:
CPU STRUCTURE AND FUNCTION

Processor Organisation

Hello friends! Today I am going to discuss with you about the aspects of the processor , iwill begin with summary of processor organization.Registers,which form the internal memory of the processor,along with it we would discuss the instruction cycle and the common technique known as instruction pipelining .So let us beign …

To understand the organisation of the CPU, let us consider the requirements placed on the CPU, the things that it must do :

- **Fetch Instructions**: The CPU must read instructions from Memory.
- **Interpret Instructions**: The instruction must have be decoded to determine what action is required.
- **Fetch data**: The execution of an instruction may require reading from memory or I/O module.
- **Process data**: The execution of an instruction may require performing some arithmetic or logical operation on data.
- **Write Data**: The results of an execution may require writing data to memory or an I/O module.

In order to be able to do these things, it should be clear that the CPU needs to temporarily store some data. It must remember the location of the last instruction so that it can know where to get the next instruction. It needs to store instructions and data temporarily while an instruction is being executed. In other words, the CPU needs is a same internal memory.

Fig. 24.1 is a simplified view of a CPU, indicating its connection to the rest of the system via the system bus. A similar interface would be needed for any of the interconnection structures described in chapter 3. The reader will recall that the major components of the CPU are an Arithmetic and Logic unit (ALU) and a control unit.The ALU does the actual computation or processing of data.The control unit controls the movement of data and instructions into and out of the CPU and controls the operation of the ALU. In addition, the figure shows a minimal internal memory, consisting of a set of storage locations, called registers.

Figure 24.2 is a slightly more detailed view of the CPU. The data transfer and logic control paths are indicated, including an element labeled internal CPU bus. This element is needed to transfer data between the various resisters and the ALU, since the ALU infact operators only one an data in the internal CPU memory. The figure also shows typical basic elements of the ALU. Note the similarly Between the internal structure of the computer as whole and the internal structure of the CPU. In both cases; there is a small collection of major elements (Computer CPU, I/O, Memory ; CPU : Control unit , ALU, register) connected by data paths.

Register Organization

As we discussed in Chapter 4, a computer system employs a memory hierarchy At higher levels of the hierarchy, memory is faster, smaller and more expensive (Per bit) Within the CPU, there is a set of registers that function as a level of memory

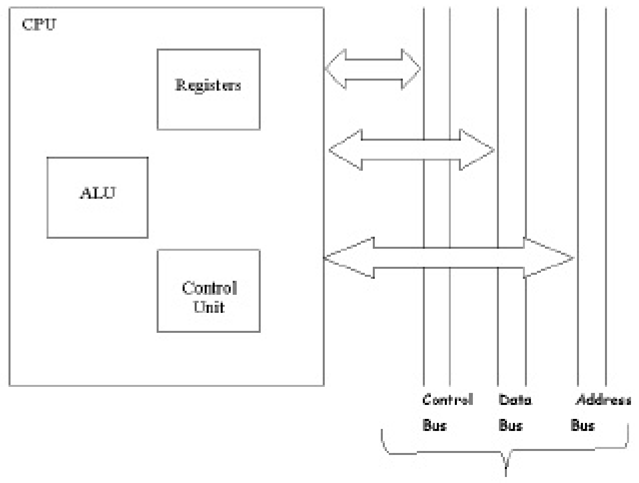


Figure 24.1The CPU with the system bus.

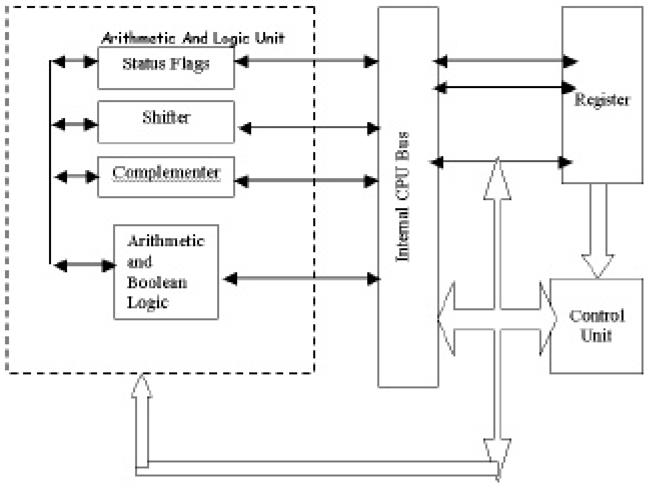


Fig.24.2 Internal structure of the CPU

Above main memory and cache in the hierarchy. The registers in the CPU serve two functions :

- User – Visible Registers : These enable the machine or assembly language programmer to minimize main memory references by optimizing use of registers.
- Control and status Registers : These are used by the control unit to control the operation of the CPU and by privileged,

operating system programs to control the execution of programs.

There is not a clean separation of register into these two categories. For example, on some machines the program counter is user visible (e.g. VAX) but on many it is not. For purpose of the following discussion., however we will use these categories.

User – Visible Register

A user-visible register is one that may be referenced by means of the machine language that the CPU executes. Virtually all contemporary CPU designs provide for a number of user visible register, as opposed to a single accumulator. We can characterize these in the followings categories :

- General Purpose
- Data
- Address
- Condition Codes

General Purpose registers can be assigned to a verity of functions by the programmer. Sometimes, their use within the instruction set is orthogonal to the operation. That is nay general- purpose register can contain the operand for any opcode. This provides true general purpose register use. Often, However, there are restrictions. For example, there may be dedicated registers for floating point operations.

In some cases , general purpose registers can be used for addressing functions (e.g. register indirect, displacement). In other cases, there is a partial or clean separation between data register and address register. Data register may be used only to hold data and cannot be employed in the calculation of an operand address. Address register may themselves be some-what general purpose, or they may be devoted to a particular addressing mode, example include.

- **Segment Pointers** : In a machine with segmented addressing, a segment register holds the address of the base of the segment. There may be multiple register : for example, one for the operating system and one for the current process
- **Index Registers** : There are used for indexed addressing and may be auto indexed.
- **Stack Pointer** : If there is user-visible stack addressing, then typically the stack is in memory and there is dedicated register that points to the top of the stack. This allows implicit addressing ; that is, push, pop and other stack instructions need not contain an explicit stack operand

There are several design issues to be addressed here. An important one is whether to use completely general purpose register or to specialize their use. We have already touched on this issue in the preceding chapter, since it affects instruction set design. With the use of specialized registers, it can generally be implicit in the opcode which type of register a certain operand specifier refers to. The operand specifeir must only identify one of a set of specialized registers rather than one out of all the register, thus saving bits. On the other hand, this specialization limits the programmers flexibility. There is no final and best solution to this design issue, but as was mentioned, the trend seems to be toward the use of specialized registers.

Another design issue is the number of registers, either general-purpose or data plus address, to be provided. Again, this affects instruction set design since more registers require more operand specifier bits. As we previously discussed, some where between 8 and 32 registers appears optimum [LUND77]. Fewer registers result in more memory references ; more register do not noticeably reduce memory references (e.g. see [WILL90]. However, a new approach, which finds advantage in the use of hundreds of registers, is exhibited in some RISC systems.

Finally, there is the issue of register length. Register that must hold addresses obviously must be at least long enough to hold the largest address. Data registers should be able to hold values of most data types. Some machines allow two contiguous registers to be used as one for holding double-length values.

A finally category of registers, which is at least partially visible to the user, holds condition codes (also referred to as flags). Condition codes are bits set by the CPU hardware as the result of operations. For example, an arithmetic operation may produce a positive, negative , zero or overflow result. In addition to the result itself being stored in a register or memory, a condition code is also set. The code may subsequently be tested as part of a conditional branch operation.

Condition code bits are collected into one or more registers. Usually, they form part of a control register. Generally, machine instructions allow these bits to be read by implicit reference, but they can not be altered by the programmer.

In some machines, a subordinate call will result in the automatic saving of all user visible registers, to be restored on return. The saving and restoring is performed by the CPU as part of the execution of call and return instructions. This allows each subroutine to use the user visible registers independently. On other machines, it is the responsibility of the programmer to save the contents of the relevant user visible registers prior to a call, by including instructions for this purpose in the program.

Control and Status Registers

There are a variety of CPU registers that are employed to control the operation of the CPU. Most of these, on most machines, are not visible to the user. Some of them may be visible to machine instructions executed in a control or operating system mode.

Of course, different machines will have different register organizations and use different terminology. We list here a reasonably complete list of register types, with a brief description.

Four registers are essential to instruction execution :

- **Program Counter (PC)** : Contains the address of an instruction to be fetched.
- **Instruction Register (IR)** : Contains the instruction most recently fetched.
- **Memory Address Register (MAR)** : Contains the address of a location in memory.

- **Memory Buffer Register (MBR)** : Contains a word of data to be written to memory or the word most recently read.

The Program counter contains an instruction address. Typically, the program counter is updated by the CPU after each instruction fetch so that it always points to the next instruction to be executed. A branch or skip instruction will also modify the contents of the PC. The fetched instruction is loaded into an instruction register, where the opcode and operand specifiers are analyzed. Data are exchanged with memory using the MAR and MBR. In a bus organized system, the MAR connects directly to the address bus, and the MBR connects directly to the data bus. User Visible registers, in turn, exchange data with the MBR.

The four registers just mentioned are used for the movement of data between the CPU and memory. Within the CPU, data must be presented to the ALU for processing. The ALU may have direct access to the MBR and user visible registers. Alternatively, there may be have additional buffering registers at the boundary to the ALU; these registers serve as input and output registers for the ALU and exchange data with the MBR and user-visible registers.

The CPU design include a register or set of registers, often known as the program status word (PSW), that contain status information. The PSW typically contains condition codes plus other status information. Common fields or flags include the following :

- **Sign** : Contains the sign bit of the result of the last arithmetic operation.
- **Zero** : Set when the result if 0.
- **Carry** : Set if an operation resulted in a carry (addition) into or borrow (subtraction) out of a high order bit. Used for multiword arithmetic operations
- **Equal** : Set if a logical compare result a equality.
- Overflow : Used to indicate arithmetic overflow.
- Interrupt Enable / Disable : Used to enable or disable interrupts.
- **Supervisor** : Indicates whether the CPU is executing in supervisor or user mode. Certain privileged instructions can be executed only one in supervisor mode, and certain areas of memory can be accessed only in supervisor mode, and certain areas of memory can be accessed only in supervisor mode.

There are a number of other registers related to status and control that might be found in a particular CPU Design. In addition to the PSW, there may be a pointer to a block of memory containing additional status information (e,g. process control blocks). In machines using vectored interrupts, an interrupts vector register may be provided . If a stack is used to implement certain functions (e.g. subroutine call), then a system stack pointer is needed. A Page table pointer is used with a virtual memory system. Finally, registers may be used in the control of I/O operations.

A number of factors go into the design of the control and status register organization. One key issue is operating system support. Certain types of control information are of specific utility to the operating system. If the CPU designer has a functional understanding of the operating system to be used, then the register organizations can to some extent be tailored to the operating system.

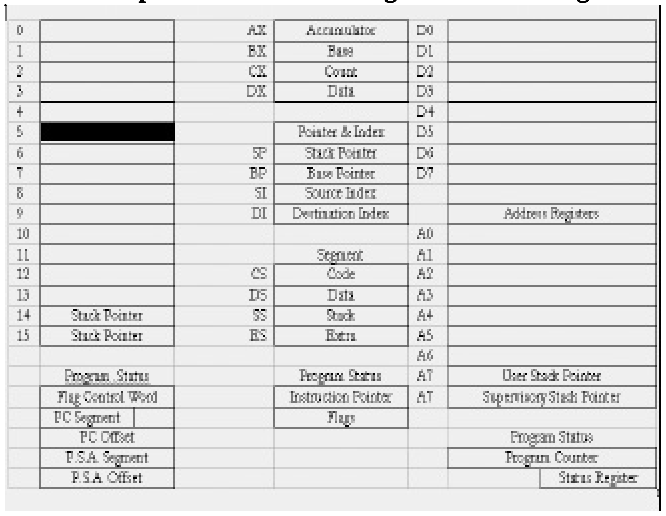
Another key design decision is the allocation of control information between registers and Memory. It is common to dedicate the first [lowest] few hundred or thousand words of memory for control purposes. The designer must decide how much control information should be in registers and how much in memory . The usual trade –off of cost versus speed arises .

Example Microprocessor Register Organizations

It is instructive to examine and compare the registers organisation of comparable systems. In this section, we look at three 16-Bit microprocessors that were designed at about the same time : the Zilog Z8000 [PEUT79], the Intel 8086 [MORS78, HEYW83], and the Motorola MC68000 [STRI79]. Following fig depicts the register organisation of each; purely internal registers, such as a memory address register are not shown.

The Z8000 makes use of 16-16 bit general-purpose registers, which can be used for data, addresses, and indexing. The designers felt that it was more important to provide a regularized, general set of registers than to save instruction bits by using special-purpose registers. Further, they preferred to leave it to the programmer to assign functions to registers, assuming that there might be different functional

General Purpose **General Registers** **Data Registers**



(a) Z8000 (b) 8086 (c) MC68000

Fig 24.3 Microprocessor Register Organisation.

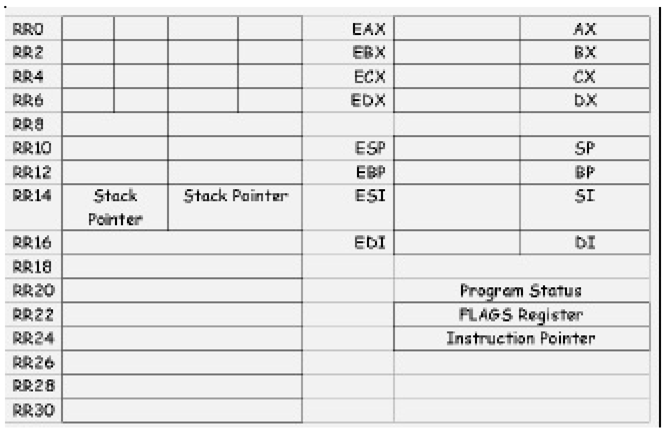
breakdowns for different applications. The registers can also be used for 8 bit and 32-bit operations. A segmented address space is used (7 bit segment number, 16- bit offset), and two registers are needed to hold a single address. To of the registers are also used as implied stack pointer for system mode and normal mode.

The Z8000 also includes five registers related to program status. Two registers hold the program counter and two the address of a program status Area in memory. A 16 bit flag register holds various status and control bits.

The Intel 8086 takes a different approach to register organization. Every register is special purpose, although some registers are also usable general purpose. The 8086 contains four 16 bit data registers that are addressable on a byte or 16 bit basis and four 16 bit pointer and index registers. The data registers can be used as general purpose in some instructions./ In others, the registers are used implicitly. Fro example, a multiply instruction Always uses the accumulator. The four pointer registers are also used implicitly in a number of operations; each contains a segment offset. There are also four 16 bit segment registers. Three of the four segment registers are used in a dedicated, implicit fashion, to point the segment of the current instruction (useful for branch instructions), a segment containing data, and a segment containing a stack, respectively. These dedicated and implicit uses provide for compact encoding at the cost of reduced flexibility. The 8086 also includes an instruction pointer and a set of 1 bit status and control flags.

The Motorola MC68000 falls somewhere between the design philosophies of the Zilog and Intel microprocessors. The MC68000 partitions its 32-bit registers into

General-Purpose Registers **General Registers**



(a) Z80,000 (b) 80386

Fig : Register Organization extensions for 32-bit microprocessors.

Eight data registers and nine address registers. The eight data registers are used primarily for data manipulation and are used in addressing only as index registers. The width of the registers allows 8-, 16- and 32 bit data operations, determined by opcode. The address contains 32-bit (no segmentation) addresses ; two of these registers are also used as stack pointers, one for users and one for the operating system. Depending on the current execution mode. Both registers are numbered 7, since only one can be used at a time. The MC68000 also includes a 32 bit program counter and a 16 bit status register.

Like the Zilog designers, the Motorola team wanted a very regular instruction set, with no special purpose registers. A concern for code efficiency led them to divide the registers into two functional components, saving one bit on each registers speicfier. This seems a reasonable compromise between complete generality and code compaction.

The Point of this comparison should be clear. There is, as yet, no universally accepted philosophy concerning the best way to organize CPU registers [TOON81]. As with overall instruction set design and so many other COU design issues, it is still a matter of judgment and taste.

A second instructive point concerning register organisation design is illustrated in Figure 11.4. this figure shows the user visible register organization for the Zilog 80,000 [PHIL85] and the Intel 80386 [ELAY85], which are 32-bit microprocessors designed as extensions of the Z8000 and 8086, respectively. Both of these new processors use 32 bit registers. However to provide upward compatibility for programs written on the earlier machines, both of the new processors retain the original register organisation embedded in the new organisation. Given this design constraint, the architects of the 32-bit processors had limited flexibility in designing the register organisation.

The Lectures to Go On!!!!!

1. Processor organization
2. Register organization
3. Types of registers

Question:

1. Explain the Different types of organization?

References:

1. Digital Logic and Computer Design—Moris Mano —Prentice Hall Of India
2. Computer System Architecture —Moris Mano—”

Note: