Stallings 10th edition

CS 320
Ch. 20 The Control Unit

Instructions are broken down into fetch, indirect, execute, and interrupt cycles. Each of these cycles, in turn, can be broken down into microoperations where a microoperation is an "atomic" operation of the processor typically done in hardware or firmware.

There are four registers involved in the fetch cycle: **MAR, MBR, PC, and IR.**

The fetch cycle can typically be broken down into four microoperations: **MAR ← PC, MBR ← M, PC ← PC + 1, IR ← MBR.**

Note that the PC is incremented in the fetch cycle. **Instructions are stored in sequence so PC is set to point to next instruction.**

The PC ← PC + 1 microoperation is somewhat independent of the others in the sequence. This means that it can be done in parallel with some of the other operations. **Microoperations can be grouped in three time slots, t1, t2, and t3. t1 must be MAR ← PC. MBR ←M and IR ← MBR are the next two. PC ← PC + 1 can go with t2 or t3.**

Note that PC ← PC + 1 involves an addition. This is done **in hardware at the PC register or it can be done by the ALU.**

The indirect cycle is n**ot needed by all instructions. This cycle is to fetch additional operands from memory.**

The microoperations in the indirect cycle are:
t1: MAR ← (IR(Address))
t2: MBR ← Memory
t3: IR(Address) ← (MBR(Address)).

The interrupt cycle begins **at the end, after the instruction has been complete and the registers are stable.**

At the end of the instruction cycle the hardware polls the interrupt bit to see if there is an interrupt.

The microoperations for the interrupt cycle are:
t1: MBR ← (PC)
t2: MAR ← Save_Address, PC ← ISR_Address
t3: Memory ← (MBR).

The Save_Address t**ypically from the Stack Pointer. This may involve multiple steps in which the stack pointer registers is altered.**

The execute cycle different from the previous three cycles in that i**t performs a different set of microoperations depending on the on the op code. The others are mostly uniform.**

15. For a simple instruction such as ADD R1, x, the microoperations in the instruction cycle are:

t1: MAR ← (IR(address))
t2: MBR ← Memory
t3: R1 ← (R1) + (MBR).

In this sequence (IR(Address)) is t**he address in the address field of the instruction register.**

Consider the following sequence of microoperations for the Increment and Skip if Zero instruction. ISZ x.

t1: MAR ← (IR(address))
t2: MBR ← Memory
t3: MBR ← (MBR) + 1
t4: Memory ← (MBR)
If ((MBR) == 0) then (PC ← PC + 1)

To do the if statement in hardware as a microoperation the b**its of MBR go to a NAND gate. If the MBR has all zeros then the NAND gate will output a one. This one is ANDed with a clock pulse to the PC.**

See Figure 19.3 below:

The block marked ICC is the **Instruction Cycle Code 00 = fetch, 01 = indirect, 10 = execute, and 11 = interrupt.**

The indirect cycle seems to be very simple. **It puts the indirect address into the IR address field thereby making it a direct address. It then goes directly to the execute cycle.**

The block marked Opcode has many lines coming out of it that go nowhere. **That part of the flow chart is dependent on the op code.**

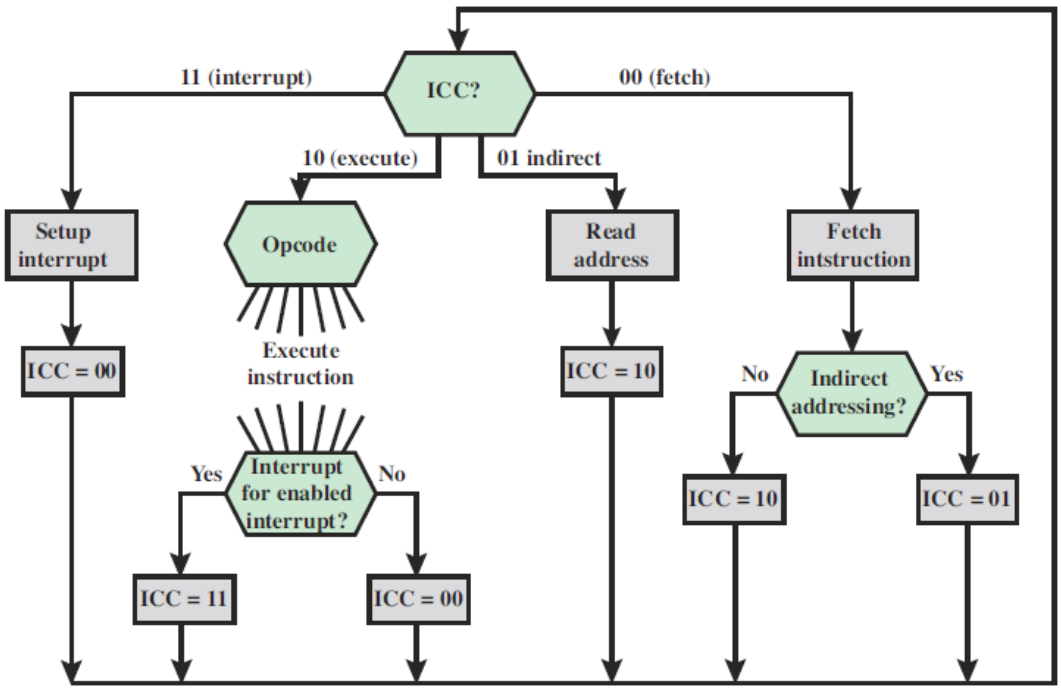


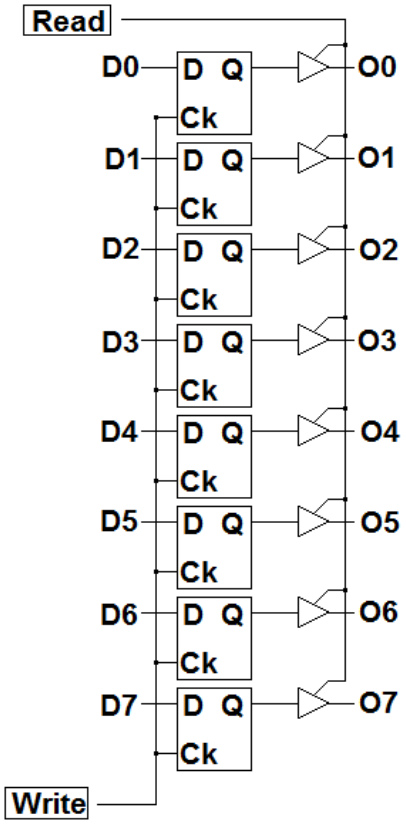
Figure 19.3 Flowchart for Instruction Cycle

All microoperations fall into one of four categories.

A) Transfer data between registers
B) Transfer data from a register to an external bus
C) Transfer data from an external bus to a register
D) Perform arithmetic or logical operations using registers for input and output.

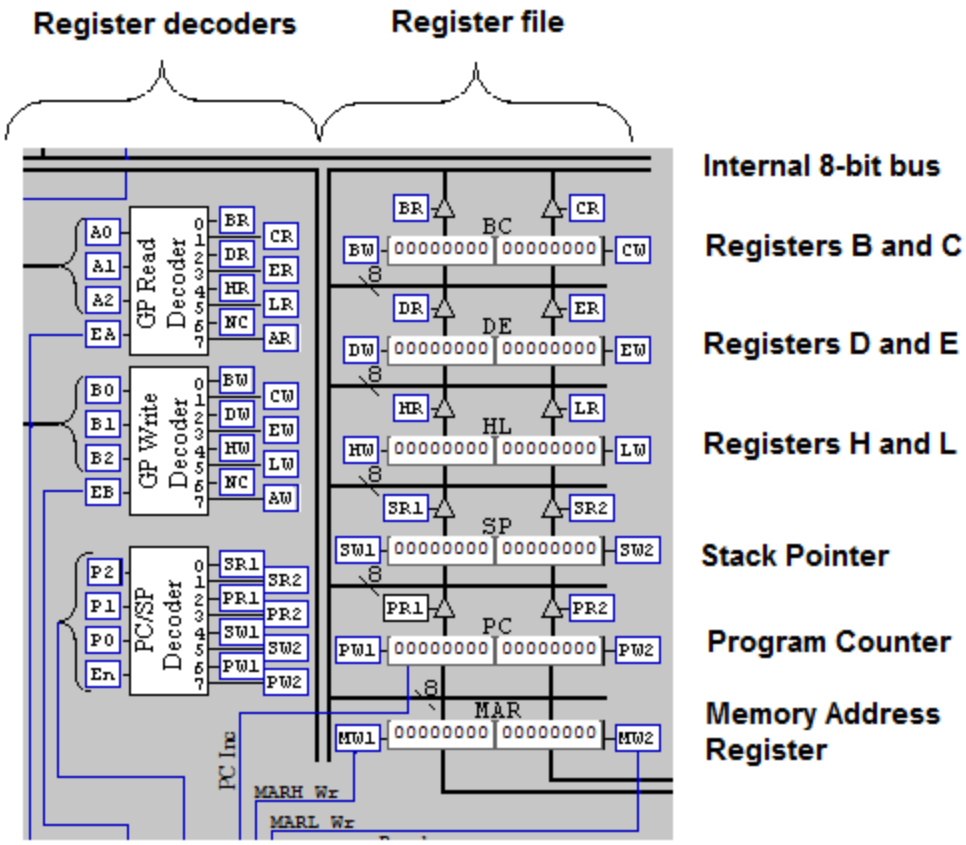
In hardware a register consists of a **collection of flip-flops with a common clock. Each flip-flop input connects to a bus line and each output connects to a bus line via a tri-state gate. The tri-state control is common for all flip-flops.**

Stallings 10th edition



A microoperation transfers data from one register to another by **setting the tri-state control of one register to allow its output to get on the bus and it sets the clock of the other to clock the data from the bus.**

Stallings 10th edition



The control unit outputs microoperations in a proper time sequence. This is generally referred to as sequencing and the control unit may be called a sequencer.

The inputs to the control unit are the **clock, IR, flags, control signals from the control bus.**

The flags **are the condition codes set by ALU ops. They allow the control unit to do such things as conditional branching. Effectively the flags hold a one-bit history of past operations.**

The control unit need to have an input from the IR: **This is the op code that determines what operation the control unit is to perform.**

Control bus lines might include: **Interrupt line, memory busy.**

The outputs from the control unit are c**ontrol signals to units within the CPU, and Control signals to the control bus.**

Control signals go from the control unit to the control bus are: **Memory read, memory write, I/O read write, interrupt acknowledge, etc.**

An example of control signals that go from the control unit to inside the CPU include l**ines to the ALU, lines to transfer data between registers, lines to test flag bits.**

See Figure 19.5 below:

The control points on say the IR (C_4 and C_{13}) are: **C_4 is the common clock signal on the IR register and C_{13} is the signal to the tri-state gates.**

The control signals for the fetch sequence:

t1: MAR ← (PC) **C_2**

t2: MBR ← Memory

PC ← (PC) + 1 **C_5, C_R (C_R is Memory read)**

t3: IR ← (MBR) **C_4**

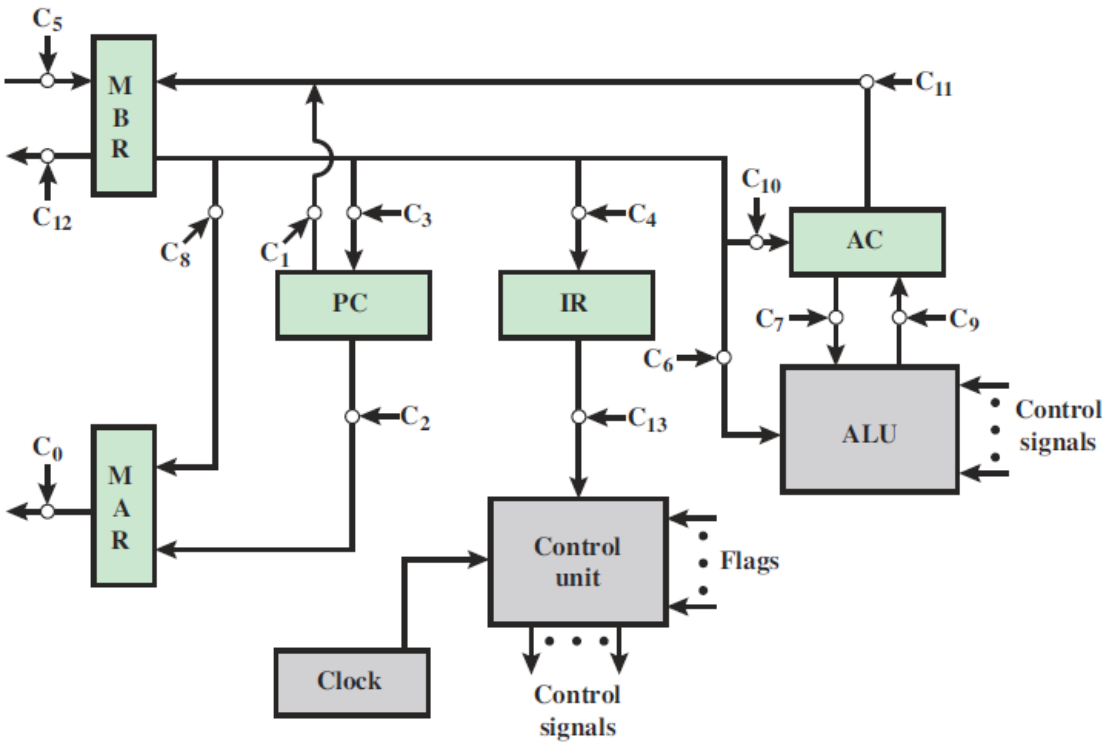


Figure 19.5 Data Paths and Control Signals

The Intel 8085

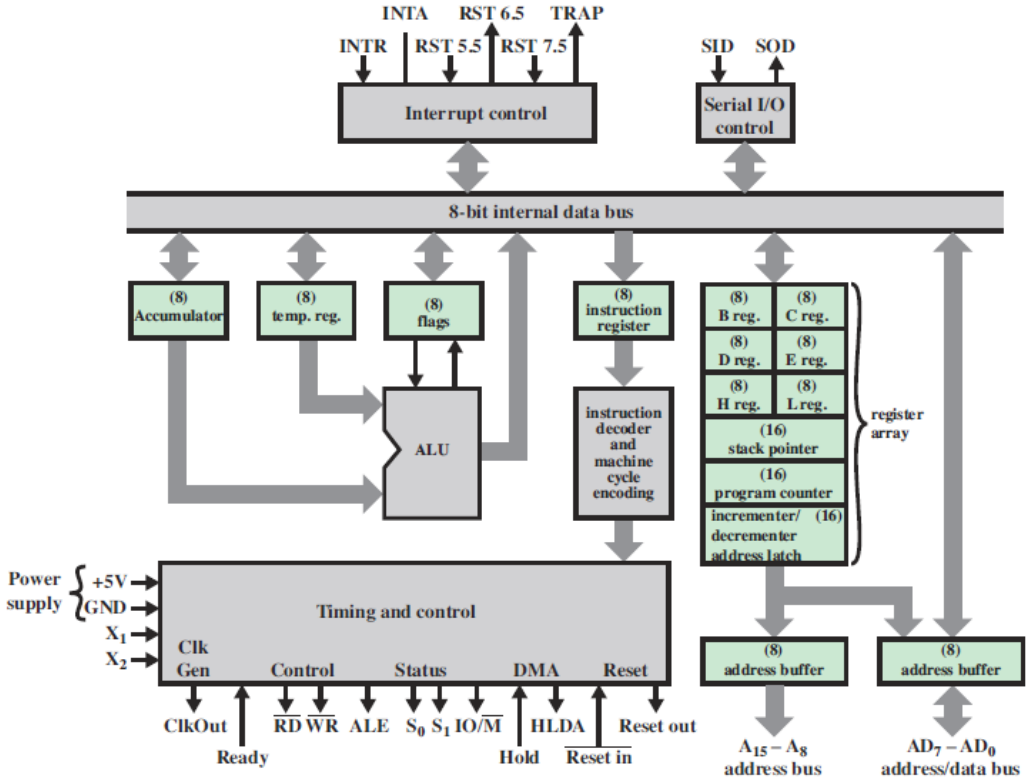
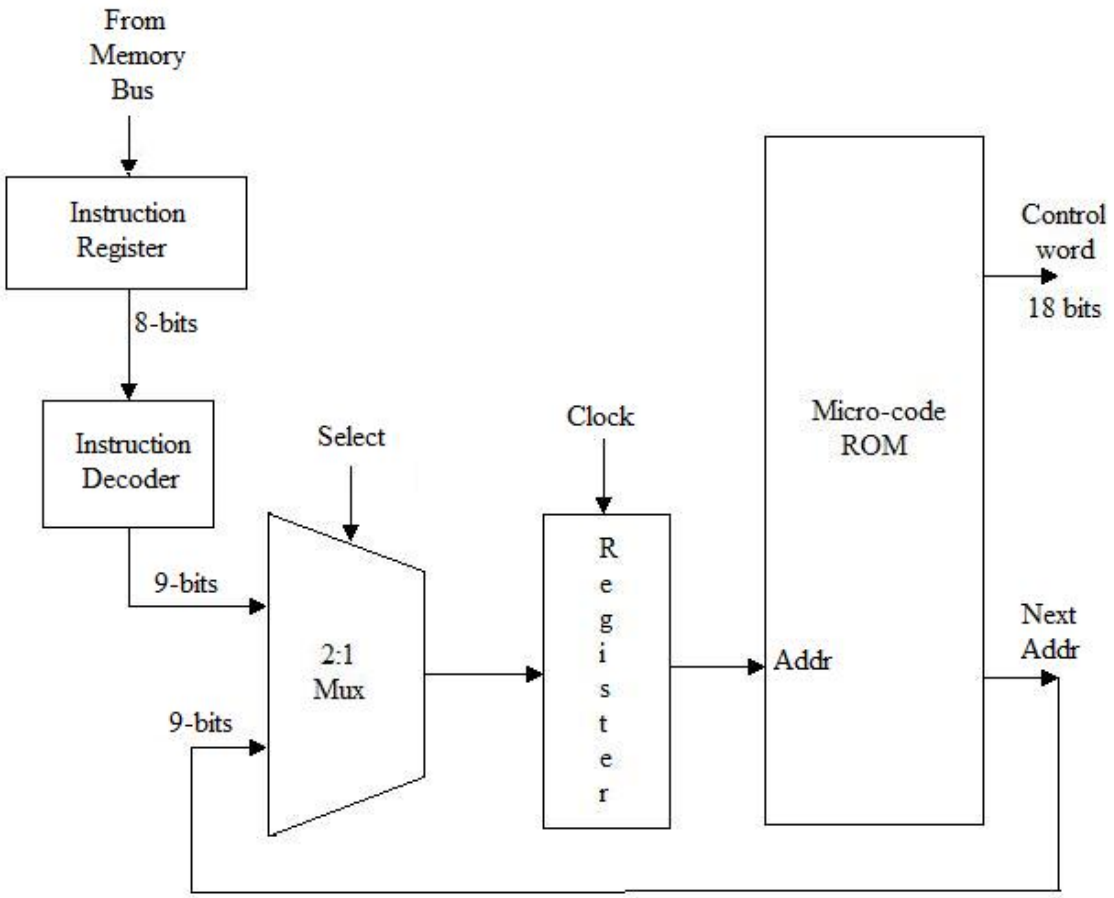


Figure 19.7 Intel 8085 CPU Block Diagram

Hard wired control **means that the controller is implemented without software. The controller is a sequential machine.**

Microprogrammed control means that the control words are stored in a read only memory and accessed as needed for control of the processor.

Stallings 10th edition



Stallings 10th edition