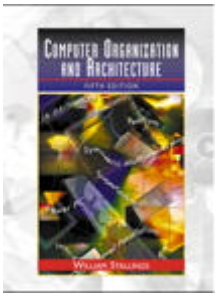
Campus de Gualtar
4710-057 Braga

UNIVERSIDADE DO MINHO
ESCOLA DE ENGENHARIA

Departamento de
Informática



Computer Organization and Architecture

5th Edition, 2000

by William Stallings

Table of Contents

I. OVERVIEW.

1. Introduction.
2. Computer Evolution and Performance.

II. THE COMPUTER SYSTEM.

3. System Buses.
4. Internal Memory.
5. External Memory.
6. Input/Output.
7. Operating System Support.

III. THE CENTRAL PROCESSING UNIT.

8. Computer Arithmetic.
9. Instruction Sets: Characteristics and Functions.
10. Instruction Sets: Addressing Modes and Formats.
11. CPU Structure and Function.
12. Reduced Instruction Set Computers (RISCs).
13. Instruction-Level Parallelism and Superscalar Processors.

IV. THE CONTROL UNIT.

14. Control Unit Operation.
15. Microprogrammed Control.

V. PARALLEL ORGANIZATION.

16. Parallel Processing.

Appendix A: Digital Logic.
Appendix B: Projects for Teaching Computer Organization and Architecture.
References.
Glossary.
Index.
Acronyms.

III. THE CENTRAL PROCESSING UNIT.

8. ...
9. ...
10. ...
11. CPU Structure and Function. (29-Apr-98)

Processor Organization (11.1)

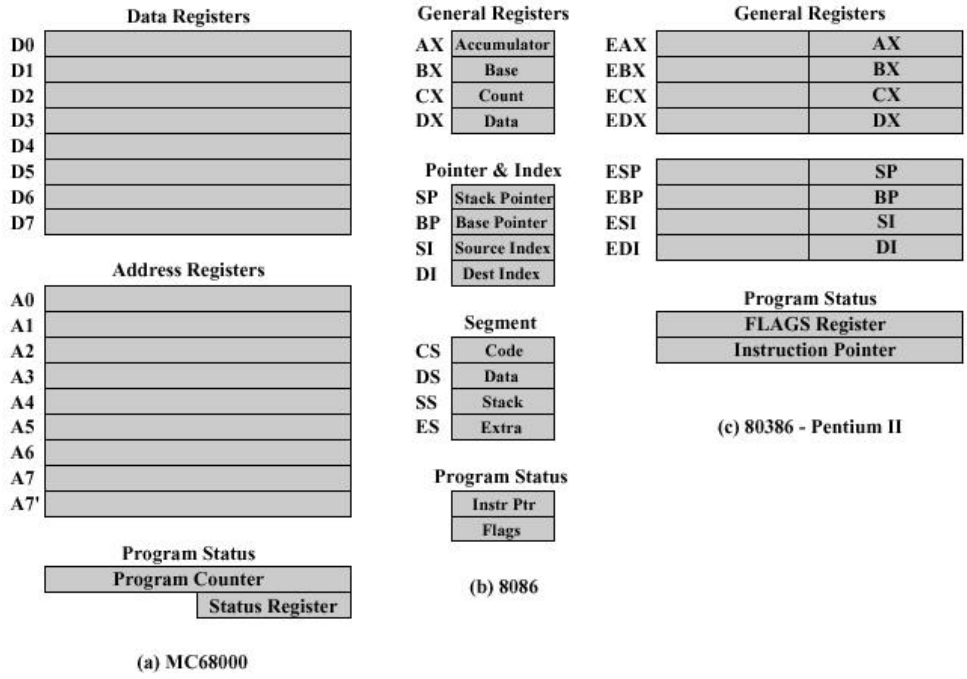
- Things a CPU must do:
    - Fetch Instructions
    - Interpret Instructions
    - Fetch Data
    - Process Data
    - Write Data
- A small amount of internal memory, called the registers, is needed by the CPU to fulfill these requirements

Register Organization (11.2)

- Registers are at top of the memory hierarchy. They serve two functions:
    - User-Visible Registers - enable the machine- or assembly-language programmer to minimize main-memory references by optimizing use of registers
    - Control and Status Registers - used by the control unit to control the operation of the CPU and by privileged, OS programs to control the execution of programs
- User-Visible Registers
    - Categories of Use
        - General Purpose
        - Data
        - Address
            - Segment pointers - hold base address of the segment in use
            - Index registers - used for indexed addressing and may be auto indexed
            - Stack Pointer - a dedicated register that points to top of a stack. Push, pop, and other stack instructions need not contain an explicit stack operand.
        - Condition Codes
    - Design Issues
        - Completely general-purpose registers, or specialized use?
            - Specialized registers save bits in instructions because their use can be implicit
            - General-purpose registers are more flexible
            - Trend is toward use of specialized registers
        - Number of registers provided?
            - More registers require more operand specifier bits in instructions
            - 8 to 32 registers appears optimum (RISC systems use hundreds, but are a completely different approach)
        - Register Length?
            - Address registers must be 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 for double-length values

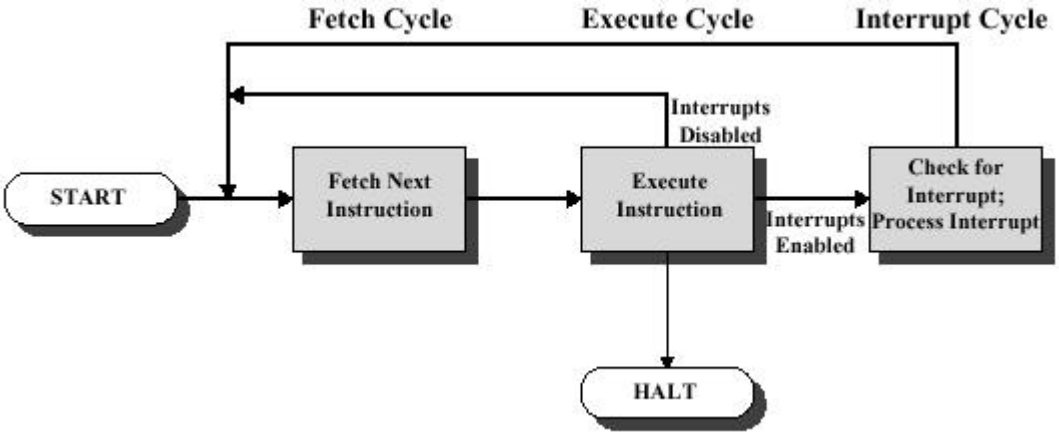
        - Automatic or manual save of condition codes?
            - Condition restore is usually automatic upon call return
            - Saving condition code registers may be automatic upon call instruction, or may be manual

- Control and Status Registers
    - Essential to instruction execution
        - Program Counter (PC)
        - Instruction Register (IR)
        - Memory Address Register (MAR) - usually connected directly to address lines of bus
        - Memory Buffer Register (MBR) - usually connected directly to data lines of bus
    - Program Status Word (PSW) - also essential, common fields or flags contained include:
        - Sign - sign bit of last arithmetic op
        - Zero - set when result of last arithmetic op is 0
        - Carry - set if last op resulted in a carry into or borrow out of a high-order bit
        - Equal - set if a logical compare result is equality
        - Overflow - set when last arithmetic operation caused overflow
        - Interrupt Enable/Disable - used to enable or disable interrupts
        - Supervisor - indicates if privileged ops can be used
    - Other optional registers
        - Pointer to a block of memory containing additional status info (like process control blocks)
        - An interrupt vector
        - A system stack pointer
        - A page table pointer
        - I/O registers
    - Design issues
        - Operating system support in CPU
        - How to divide allocation of control information between CPU registers and first part of main memory (usual tradeoffs apply)
- Example Microprocessor Register Organization



The Instruction Cycle (11.3)

- Review: Basic instruction cycle contains the following sub-cycles (some repeated)



  - Fetch - read next instruction from memory into CPU
  - Execute - Interpret the opcode and perform the indicated operation
  - Interrupt - if interrupts are enabled and one has occurred, save the current process state and service the interrupt
- The Indirect Cycle
  - Think of as another instruction sub-cycle
  - May require just another fetch (based upon last fetch)
  - Might also require arithmetic, like indexing
- Data Flow
  - Exact sequence depends on CPU design
  - We can indicate sequence in general terms, assuming CPU employs:
    - a memory address register (MAR)
    - a memory buffer register (MBR)
    - a program counter (PC)
    - an instruction register (IR)
- Fetch cycle data flow
  - PC contains address of next instruction to be fetched
  - This address is moved to MAR and placed on address bus
  - Control unit requests a memory read
  - Result is
    - placed on data bus
    - result copied to MBR
    - then moved to IR
  - Meanwhile, PC is incremented
- Indirect cycle data flow
  - After fetch, control unit examines IR to see if indirect addressing is being used. If so:
  - Rightmost n bits of MBR (the memory reference) are transferred to MAR
  - Control unit requests a memory read, to get the desired operand address into the MBR
- Instruction cycle data flow

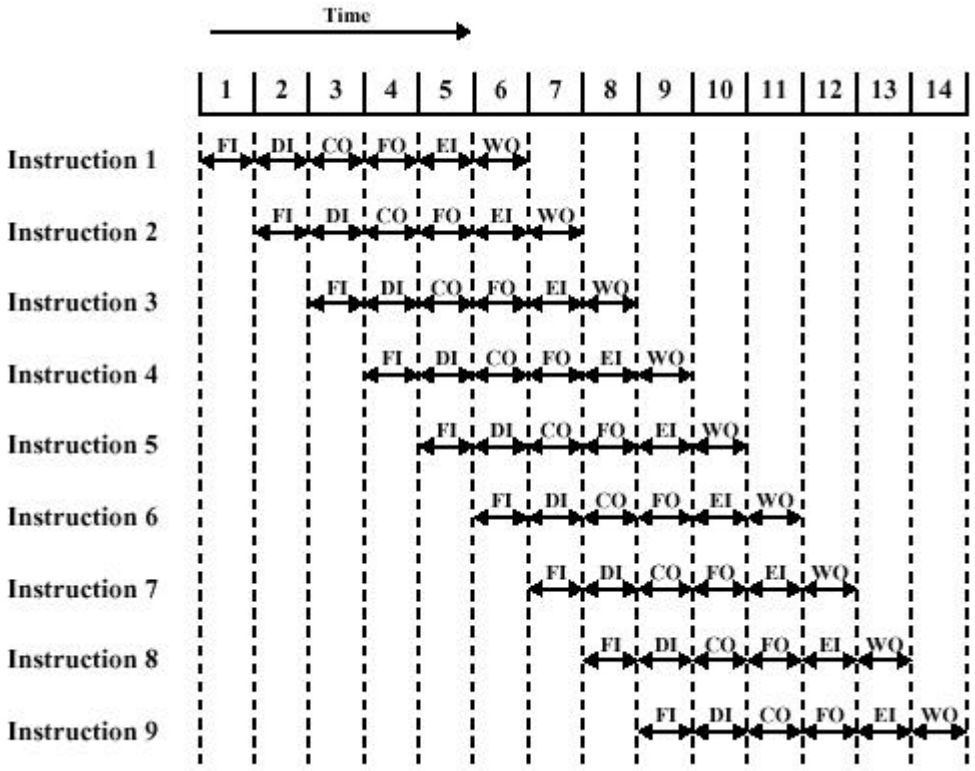
  - Not simple and predictable, like other cycles
  - Takes many forms, since form depends on which of the various machine instructions is in the IR
  - May involve
    - transferring data among registers
    - read or write from memory or I/O
    - invocation of the ALU

- Interrupt cycle data flow
  - Current contents of PC must be saved (for resume after interrupt), so PC is transferred to MBR to be written to memory
  - Save location's address (such as a stack ptr) is loaded into MAR from the control unit
  - PC is loaded with address of interrupt routine (so next instruction cycle will begin by fetching appropriate instruction)

Instruction Pipelining (11.4)

- Concept is similar to a manufacturing assembly line
  - Products at various stages can be worked on simultaneously
  - Also referred to as pipelining, because, as in a pipeline, new inputs are accepted at one end before previously accepted inputs appear as outputs at the other end
- Consider subdividing instruction processing into two stages:
  - Fetch instruction
  - Execute instruction
- During execution, there are times when main memory is not being accessed.
- During this time, the next instruction could be fetched and buffered (called instruction prefetch or fetch overlap).
- If the Fetch and Execute stages were of equal duration, the instruction cycle time would be halved.
- However, doubling of execution time is unlikely because:
  - Execution time is generally longer than fetch time (it will also involve reading and storing operands, in addition to operation execution)
  - A conditional branch makes the address of the next instruction to be fetched unknown (although we can minimize this problem by fetching the next sequential instruction anyway)
- To gain further speedup, the pipeline must have more stages. Consider the following decomposition of instruction processing:
  - Fetch Instruction (FI)
  - Decode Instruction (DI) - determine opcode and operand specifiers
  - Calculate Operands (CO) - calculate effective address of each source operand
  - Fetch Operands (FO)
  - Execute Instruction (EI)
  - Write Operand (WO)

- Timing diagram, assuming 6 stages of fairly equal duration and no branching



Notes on the diagram

  - Each instruction is assumed to use all six stages
    - Not always true in reality
    - To simplify pipeline hardware, timing is set up assuming all 6 stages will be used
  - It assumes that all stages can be performed in parallel
    - Not actually true, especially due to memory access conflicts
    - Pipeline hardware must accommodate exclusive use of memory access lines, so delays may occur
    - Often, the desired value will be in cache, or the FO or WO stage may be null, so pipeline will not be slowed much of the time
- If the six stages are not of equal duration, there will be some waiting involved for shorter stages
- The CO (Calculate Operands) stage may depend on the contents of a register that could be altered by a previous instruction that is still in the pipeline
- It may appear that more stages will result in even more speedup
  - There is some overhead in moving data from buffer to buffer, which increases with more stages
  - The amount of control logic for dependencies, etc. for moving from stage to stage increases exponentially as stages are added
- Conditional branch instructions and interrupts can invalidate several instruction fetches

Dealing with Branches

- A variety of approaches have been taken for dealing with conditional branches:
    - Multiple Streams
        - Instead of choosing one of the two instructions, replicate the initial portions of the pipeline and allow it to fetch both instructions, making use of two streams.
        - Problems:
        - Contention delays for access to registers and memory
        - Additional branch instructions may enter either stream of the pipeline before the original branch decision is resolved
        - Examples: IBM 370/168 and IBM 3033
    - Prefetch Branch Target
        - The target of the branch is prefetched, in addition to the instruction following the branch. The target is saved until the branch instruction is executed, so it is available without fetching at that time.
        - Example: IBM 360/91
    - Loop Buffer
        - A small, very-high-speed memory maintained by the instruction fetch stage of the pipeline
            - contains the n most recently fetched instructions, in sequence
            - if a branch is to be taken, the buffer is checked first and the next instruction fetched from it instead of memory
        - Benefits
            - It will often contain an instruction sequentially ahead of the current instruction, which can be used for prefetching
            - If a branch occurs to a target just a few locations ahead of the branch instruction, the target may already be in the buffer (especially useful for IF-THEN and IF-THEN-ELSE sequences)
            - As implied by the name, if the buffer is large enough to contain all the instructions in a loop, they will only have to be fetched from memory once for all the consecutive iterations of that loop
        - Similar in principle to an instruction cache, but
            - it only holds instructions in sequence
            - smaller and thus lower cost
        - Examples: CDC Star-100, 6600, 7600 and CRAY-1
    - Branch Prediction
        - Try to predict which branch will be taken, and prefetch that instruction
        - Static techniques
            - Predict Never Taken
                - Assume that the branch will not be taken and continue to fetch in sequence
                - Examples: Motorola 68020 and VAX 11/780
            - Predict Always Taken
                - Assume that the branch will always be taken, and always fetch the branch target
                - Studies show that conditional branches are taken more than 50% of the time

                NOTE: Prefetching the branch target is more likely to cause a page fault; so paged machines may employ an avoidance mechanism to reduce this penalty.
            - Predict by Opcode
                - Assume that the branch will be taken for certain branch opcodes and not for others
                - Studies report success rates of greater than 75%

- Dynamic Techniques
  - Taken/Not Taken Switch
    - Assume that future executions of the same branch instruction will branch the same way
    - Associate one or two extra bits with the branch instruction in high-speed memory (instruction cache or loop buffer) indicating whether it was taken the last one or two times
    - Two bits allow events like loops to only cause one wrong prediction instead of two
    - Example: IBM 3090/400
  - Branch History Table
    - Solves problem with Taken/Not Taken Switch, to wit: If decision is made to take the branch, the target instruction cannot be fetched until the target address is decoded
    - Branch History Table is a small cache memory associated with the instruction fetch stage of the pipeline. Each entry has: >> the address of a branch instruction >> some number of history bits that record the state of use of that instruction >> effective address of target instruction (already calculated) or the target instruction itself
    - Example: AMD29000

o Delayed Branch
- Sometimes code can be optimized so that branch instructions can occur later than originally specified
- Allows pipeline to stay full longer before potential flushing
- More detail in chapter 12 (RISC)

Intel 80486 Pipelining (11.5)

o Uses a 5-stage pipeline
- Fetch - instructions are prefetched into 1 of 2 16-byte prefetch buffers.
  - Buffers are filled as soon as old data is consumed by instruction decoder
  - Instructions are variable length (1-11 bytes)
  - On average, about 5 instructions are fetched with each 16-byte load
  - Independent of rest of pipeline
- Decode Stage 1
  - Opcode and addressing mode info is decoded
  - This info always occurs in first 3 bytes of instruction
- Decode Stage 2
  - Expands each opcode into control signals for the ALU
  - Computation of more complex addressing modes
- Execute
  - ALU operations
  - cache access
  - register update
- Write Back
  - May not be needed
  - Updates registers and status flags modified during Execute stage
  - If current instruction updates memory, the computed value is sent to the cache and to the bus-interface write buffers at the same time

o With 2 decode stages, can sustain a throughput of close to 1 instruction per clock cycle (complex instructions and conditional branches cause slowdown)