CPU Structure and Function

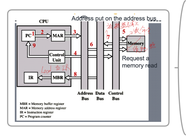
Instruction Cycle

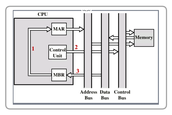
- Instruction Set Architecture
  - RISC: Reduced Set Instruction Computer
    - Simpler instructions and thus simple instruction decoding.
    - More general-purpose registers.
    - The instruction takes one clock cycle in order to get executed.
    - The instruction comes under the size of a single word.
    - Pipeline can be easily achieved.
    - Few data types.
    - Simpler addressing modes.
  - CISC : Complex Instruction Set Computer
    - Complex instructions, thus has complex instruction decoding.
    - The instructions take more than one clock cycle to get executed.
    - The instruction is larger than one-word size.
    - Lesser general-purpose registers since operations is in memory.
    - More data types.
    - Complex addressing modes
- Instruction Cycle
  - Program excution-Instruction
    -

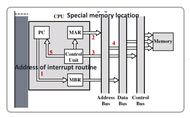
  - Instruction Cycle with Interrupts
    -

  - Instruction Cycle with Interrupts
    -

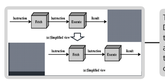
    -

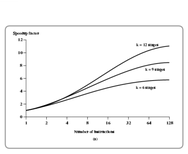
- Data Flow
  - Data Flow in Instruction Fetch
    - 1.PC contains address of next instruction
    - 2.Address moved to MAR
    - 3.Placed on address bus
    - 4.Control unit requests memory read
    - 5.Result placed on data bus, copied to MBR, then to IR
    - 6.Meanwhile PC incremented by 1
    - 

  - Data Flow (Indirect Cycle)
    - IR is examined
    - If indirect addressing, indirect cycle is performed
      - Right most N bits of MBR transferred to MAR
      - Control unit requests memory read
      - Result (address of operand) moved to MBR
    - 

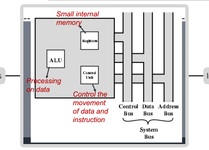
  - Data Flow in Execute Cycle
    - May take many forms. Depends on instruction being executed
  - Data Flow in Interrupt Cycle
    - Current PC saved to allow resumption after interrupt
    - Contents of PC copied to MBR
    - Special memory location (e.g. stack pointer) loaded to MAR
    - MBR written to memory
    - PC loaded with address of interrupt handling routine
    - Next instruction (first of interrupt handler) can be fetched
    - 

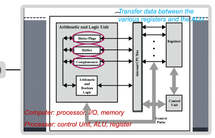
Instruction Pipelining

- New inputs are accepted at one end and transformed in outputs at the other end. Products at various stages can be worked on simultaneously, thus performance improved
- Example of a simple approach - subdivide the instruction as two stages:
  1.Fetch
  2.Execution
  - To apply this concept, an instruction must be broken down into a number of stages
- Improved Performance
  - The Fetch cycle accesses the main memory During the Execution cycle, usually, there are times when main memory is not being accessed. These times can be used to fetch the next instruction during execution of current one. This is called instruction prefetch
    - 

  - Fetch usually shorter than execution, which involves to read and the store the operand, and performs the operation.Pipeline rate limited by slowest pipeline stage
  - Conditional branch instructions make the next instruction address unknown. The fetch stage must wait until the execute stage
    - Prefetch the instruction after the branching instr.
      - If the branch is not taken, use the prefetched instr.
      - Else Discard the prefetched instruction. Fetch the new instruction
    - Multiple Streams
      - Single pipeline must choose one of two instruction in branch to fetch next and may make wrong choice
      - Have two pipelines, Prefetch each branch into a separate pipeline, Use appropriate pipeline
      - Problems
        - Leads to register & memory contention delays (which pipeline will access to memory when both of them need access at the same time ?)
        - Multiple branches lead to further pipelines being needed
    - Prefetch Branch Target
      - When a conditional branch is recognized
      - Target of branch is prefetched, in addition to instructions following the branch
      - Keep target until branch is executed
      - If the branch is executed, then the target has already been prefetched.
    - Loop buffer
      - Is a small, high speed memory
      - Similar to a cache dedicated to instructions
      - Contains the n most recently fetched instructions in sequence
      - A branch is to be taken
        - Hardware checks whether the target is in buffer
        - If YES, then next instruction is fetched from the buffer
        - Else, fetch from memory
    - Branch prediction
      - Predict whether a branch will be taken
        - If the prediction is right
          - No branch penalty
        - If the prediction is wrong
          - Empty pipeline
          - Fetch correct instruction
          - Branch penalty
      - Predict techniques
        - Static
          - Predict never taken
          - Predict always taken
          - Predict by opcode
            - Some instructions are more likely to result in a jump than others
            - Decision based on the opcode of the branch instruction.
        - Dynamic: depend on the execution history
          - Taken/not taken switch
            - One or more bits can be associated with each conditional branch instruction that reflect the recent history of the instruction.
            - Maintain a small table of recently executed branch instructions with one or more history bits in each entry
            - Direct the processor to make a particular decision when encountering it next time
          - Branch history table
  - Similar unpredictable event is an interrupt
  - To gain further speedup, add more stages to improve performance
    - More overhead in moving data between buffers, preparation and delivery functions, more total execution time
    - Data needed by current instruction may depend on a previous instruction that is still in pipeline
    - 

Processor Organization

- CPU function
  - Fetch instructions: read from memory(register, cache)
  - Interpret instructions: decode
  - Fetch data: read operand from memory or I/O
  - Process data: perform arithmetic or logical operation
  - Write data: write data to memory or I/O module
- Small internal memory in CPU
  - Store some data temporally
  - Store the context so as to start the next one
  - Store instruction and data when a instruction is executed
- CPU With Systems Bus
  - 

  - Internal Structure(ALU)
    - 

Register Organization

- Overview
  - Number and function vary between processor designs
  - Is top level of memory hierarchy
- User-visible registers(to minimize main memory reference) like ADD R1 R2 R3
  - General Purpose Registers
    - May be true general purpose
      - contain the operand for any opcode
    - May be restricted
      - dedicated for floating-point and stack operation
  - Data registers
    - only hold data
  - Address Registers
    - May devoted to a particular addressing mode or general purpose
      - Segment pointers
        - hold the base address of segment
      - Index registers
        - used for indexing addressing
      - Stack Pointer
        - point to the top of the stack
  - Condition Codes Registers(Flags)
    - Condition codes are bits set by the CPU hardware as the result of operations.
    - Arithmetic operation produce positive, zero, or overflow result.
    - At least partially visible to the user
    - Form a part of control register, can be read (implicitly) by programs,
      - e.g. Jump if zero
    - Can not (usually) be set by programmers
- Control and status registers(control) like loop
  - Overview
    - Used to control the operation of processor
    - Most are not visible to user, some are visible to instruction
  - Type
    - Program Counter(PC): next instruction (not current!)
      - PC points to next instruction to be executed, but a branch or skip instruction will modify the content of PC
    - Instruction Register(IR)
      - The instruction is loaded into IR, where the opcode and operand are analyzed
    - Memory Address Register MAR: current memory address
    - Memory Buffer Register, MBR: current data to/from memory
      - Data exchanged between MAR and MBR, MAR directly connect to the address bus, MBR directly connect to the data bus
    - Program Status Word Register : contains condition codes plus other status information
  - The design of control register
    - The support of OS, need functional understanding of the OS
    - Allocation between register and memory
- General vs Specialized
  - Make them general purpose
    - Increase flexibility and programmer options
    - BUT, Increase instruction size & complexity
  - Make them specialized
    - Smaller (faster) instructions
    - BUT, Less flexibility
    - The trend seems to be toward the use of specialized registers.
- Number of register
  - Mostly between 8 – 32
  - Fewer register results in more memory references
  - More does not noticeably reduce memory references
- Size
  - Often possible to combine two data registers to used for holding double-length values

Presented with xmind