CS257 Advanced Computer Architecture

Topic 8: CPU Control and Sequencing

Graham Martin and Matthew Leeke

{grm, matt}@dcs.warwick.ac.uk

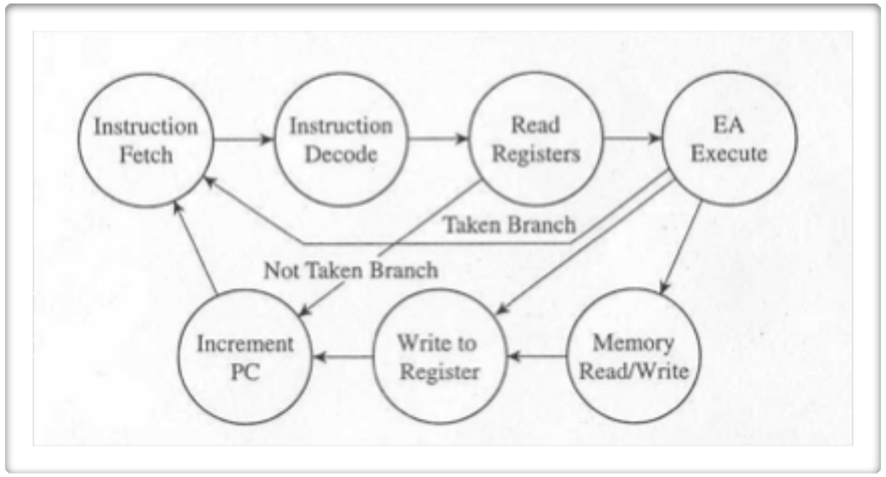
Department of Computer Science

University of Warwick

Processor Architecture

- Processor Organisation
- **CPU Control and Sequencing**
- Pipelining Fundamentals
- Superscalar Processors
- Exploiting Instruction Level Parallelism

CPU Control



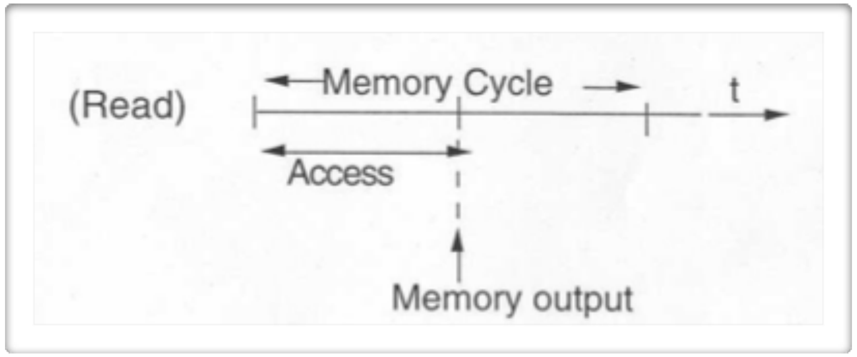
- The first step in interpretation is to fetch an instruction, pointed to by the ***program counter*** (PC)

  The instruction is decoded and, if the instructions require register values, the registers are accessed

  - The next state either computes the effective address (EA) for a memory operation or executes the function defined by the opcode
  - After sequential instruction execution the PC is incremented, and the instruction process starts again
  - As instruction fetch and memory read/write will access memory, there may be wait states for a slow memory to respond

CPU Control

- **Instruction Sequencing** (synchronous system)
  - Consider a cycle of memory activity:

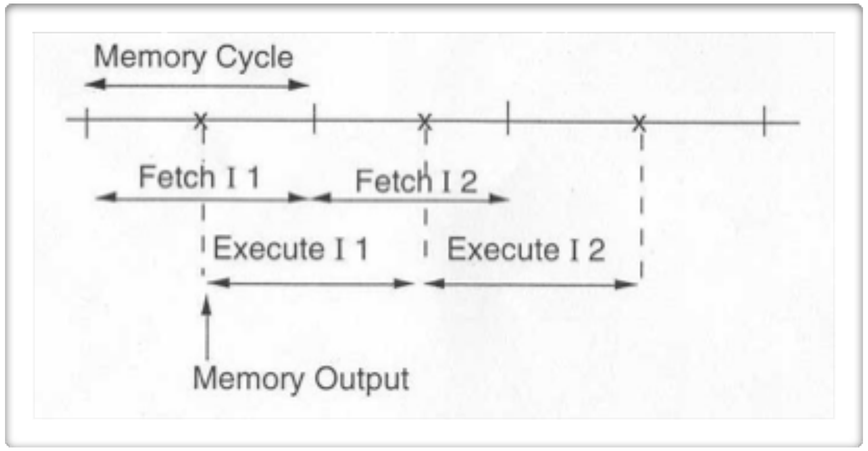


  - Design Goal: To utilise all the memory cycles

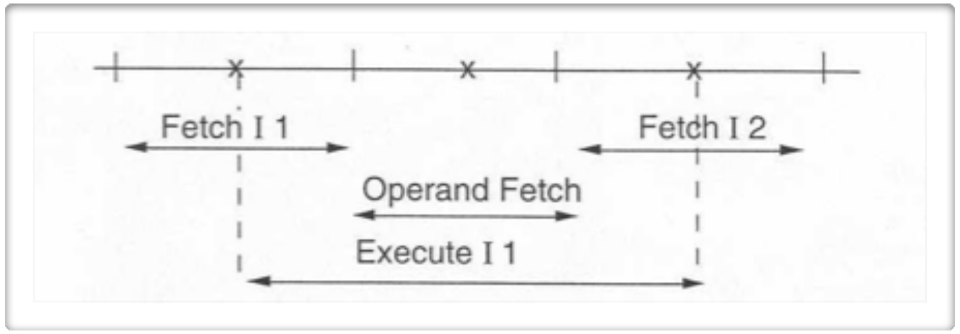
CPU Control

- Timing sequences

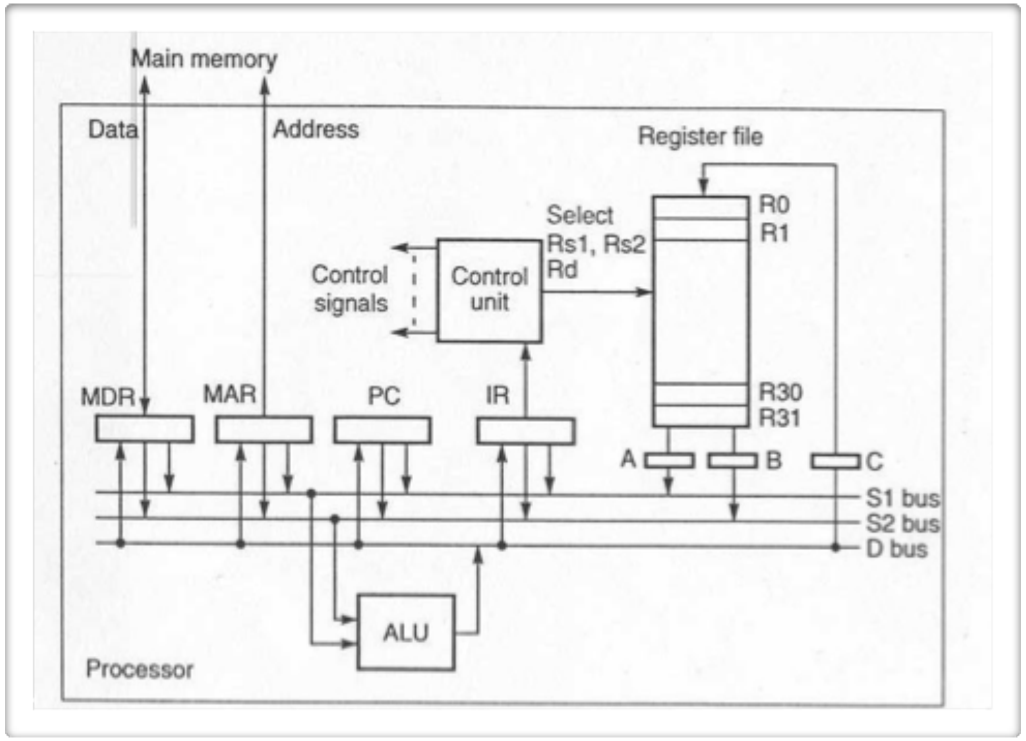
(a) Single cycle instruction



(b) Instruction requiring an operand fetch:



CPU with Centralised Control



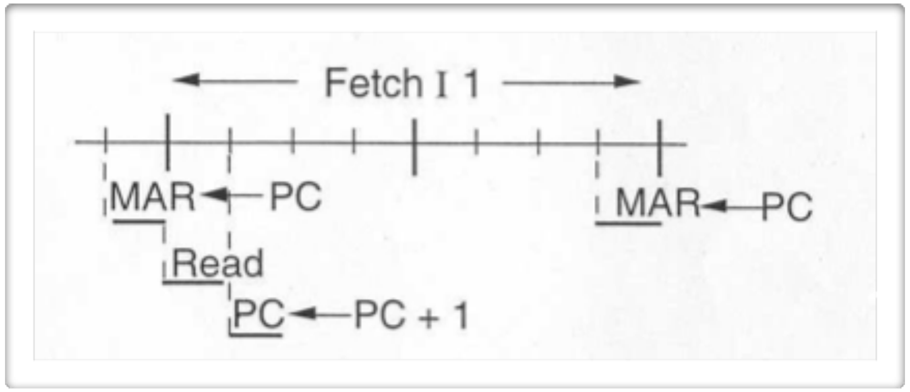
- MAR – Memory Address Register
- MDR – Memory Data Register (or MBR – Memory Buffer Register)
- IR – Instruction Register

CPU Control – Fetch Sequence

- Address of next instruction in PC is clocked into the MAR
- Address (from MAR) is placed on memory address bus
- Control unit issues memory READ command
- Result (data from memory) appears on data bus
- Data from data bus copied into MBR
- PC incremented by 1 (in parallel with data fetch from memory)
- Data (instruction) moved from MBR to IR
- MBR is now free for further data fetches

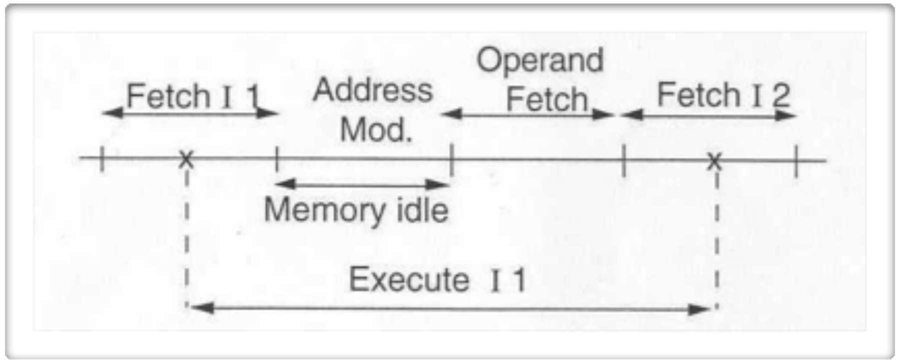
CPU Control - Rules for Clock Cycle Grouping

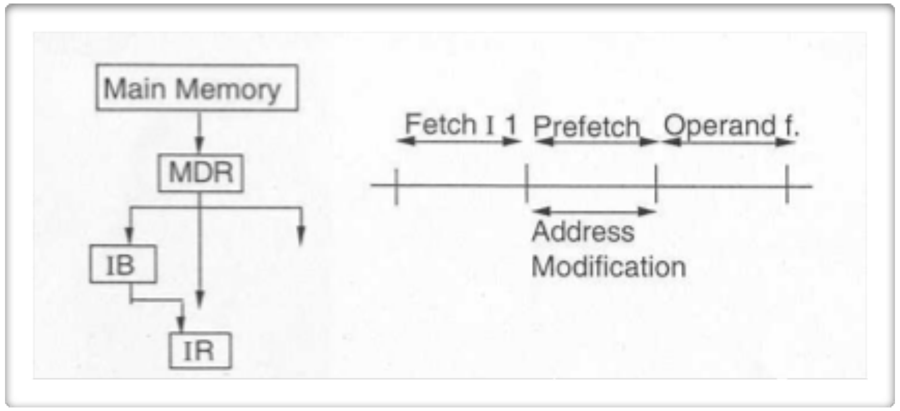
- Proper sequence must be followed
  - MAR <- (PC) must precede MBR <- (memory)
- Conflicts must be avoided
  - Must not read & write same register at same time
  - MBR <- (memory) & IR <- (MBR) must not be in same cycle
- Also: PC <- (PC) +1 involves addition
  - Use ALU, though may need additional micro-operations
- Micro-operations (micro-ops) take place during small intervals of memory cycle - defined by control and clock mechanisms



CPU Control - Instruction Prefetch

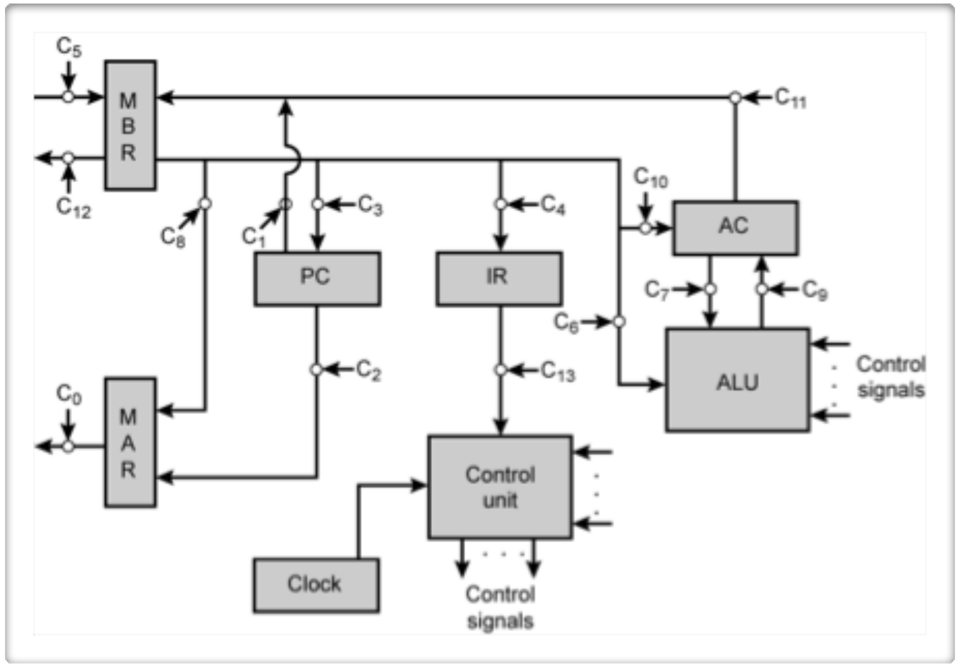
- Example: Indexed address instructions require additional gating steps and an arithmetic operation
    - this necessitates an additional memory cycle
- During the idle memory cycle, the next (sequential) instruction can be ***prefetched***, and stored in an extra instruction buffer (IB)



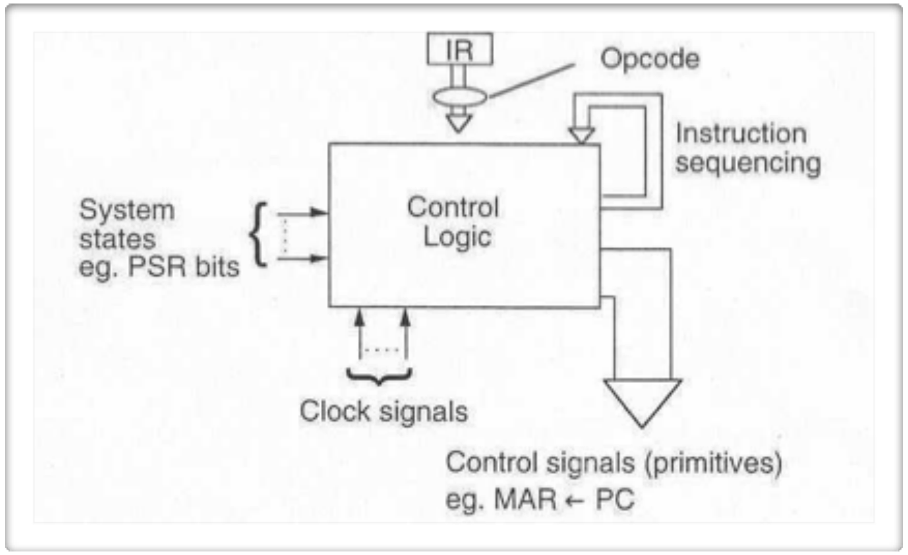


Micro-operations and Control Signals

- Micro-operations are enabled by Control Signals, required to:
  - Transfer data between registers
  - Transfer data from registers to external buses, and vice-versa
  - Perform arithmetic or logical operations



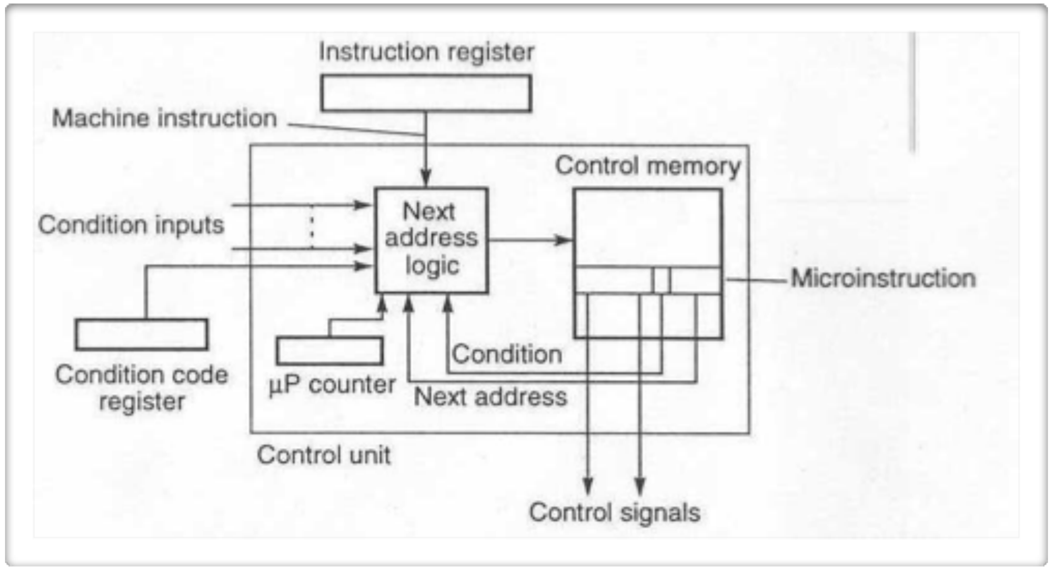
Control Signal Generation


IR
Opcode
Instruction sequencing
System states eg. PSR bits
Control Logic
Clock signals
Control signals (primitives) eg. MAR ← PC

- The control unit can be implemented using:
  - Hardwired Logic
    - Complex sequencing and micro-operation logic
    - Difficult to design and test
    - Inflexible design makes it difficult to add new instructions
  - Micro-program Store (micro-programmed control)

Micro-programmed Control

- First suggested by Wilkes (1951), but used much later
- Each step in the operation of a machine instruction is binary-encoded into a ***micro-instruction***
- Sequences of micro-instructions exist for each machine instruction.
- Micro-instructions make up the ***micro-program*** (or micro-code)



Micro-programmed Control

- Micro-program usually stored in a very fast ROM, known as the control memory
- However, the control memory can be writable - known as a ***writable control store***

  This enables the micro-code changes and allows emulation of different processors
- Operation:
  - Standard instruction fetch micro-program causes an Opcode to be read into the IR
  - Machine instruction (opcode) ‘points’ to the first microinstruction for that machine instruction
  - Sequence of microinstructions executed, providing the required generation of control signals
  - Subroutines of microinstructions often provided to reduce the size of the micro-program
  - Control ‘stack’ used to facilitate ‘return from micro-subroutine’

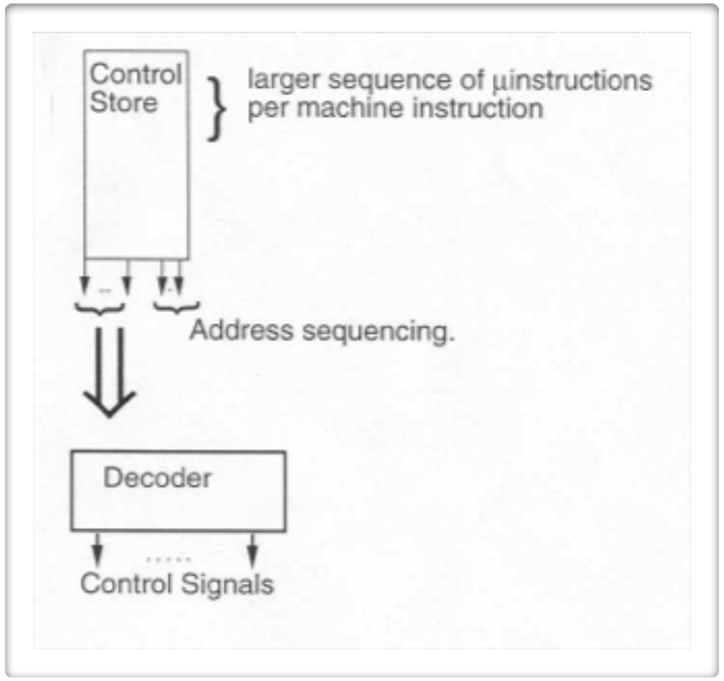
Micro-programmed Control

- The micro-program word-length is based on 3 factors:
  - Maximum number of simultaneous micro-operations supported
  - The way control information is represented or encoded
  - The way in which the next micro-instruction address is specified

- ***Horizontal or Direct Control*** (Unpacked)
  - Features a very wide microinstruction word-length, with few micro-instructions per machine instruction
  - Outputs often buffered and gated with timing signals
  - Fewer instructions, therefore increased speed

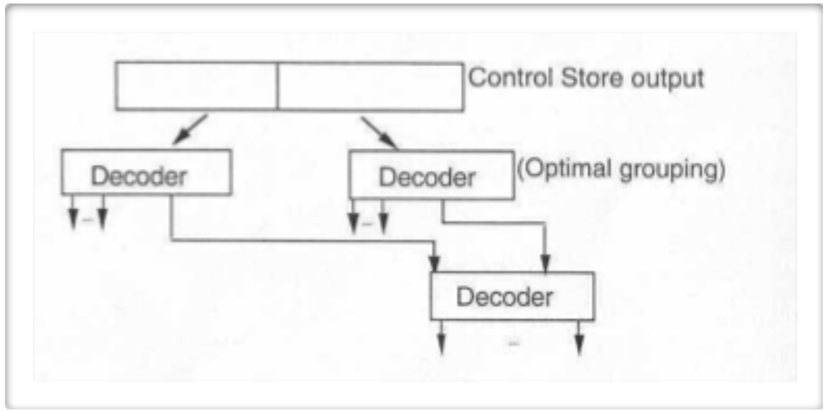
Micro-programmed Control

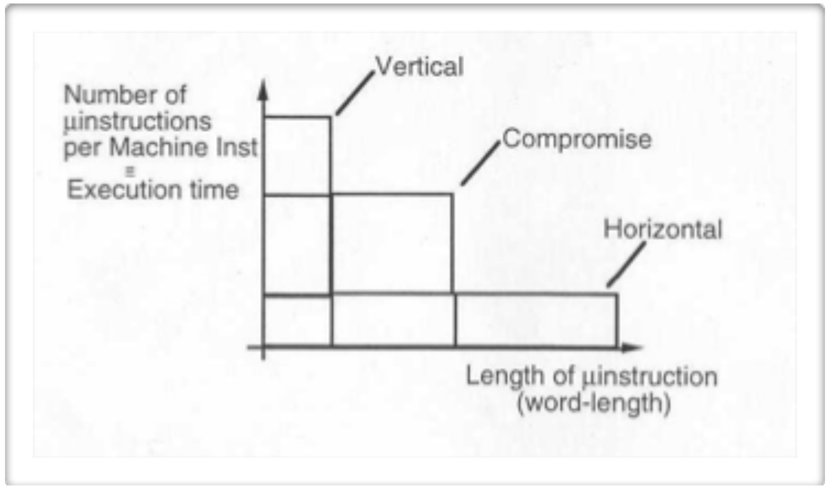
- ***Vertical Control*** (Packed / Encoded field)
  - Width is narrow
  - *n* control signals encoded into $\log_2 n$ bits
  - Limited ability to express parallelism
  - Considerable encoding of control information requires external memory word decoder to identify the exact control line being manipulated



Micro-programmed Control

- In practice, a compromise is chosen - based on cost (chip space and complexity) and performance (speed of instruction execution)
    - Divide control signals into disjoint groups
    - Implement each group as separate field in memory word
    - Supports reasonable levels of parallelism without too much complexity


Control Store output
Decoder
Decoder
(Optimal grouping)
Decoder


Number of µinstructions per Machine Inst ≡ Execution time
Vertical
Compromise
Horizontal
Length of µinstruction (word-length)