Implementation of Simple Machine With Hardwired Control

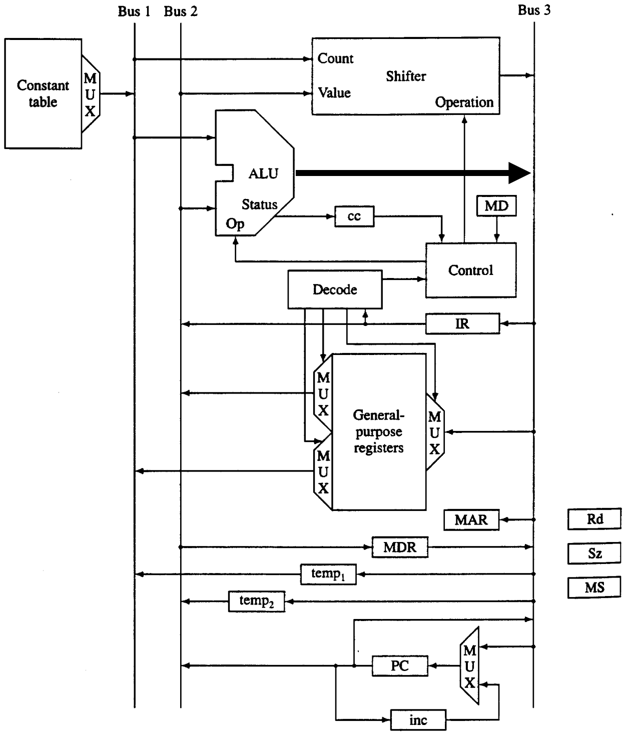

Diagram from *Computer Systems*, Maccabe, Irwin 1993

Interpreting a C Program

- Suppose we want to compile and run a program written in a *high-level programming language* (e.g.,C, C++)
  - The C compiler translates each high-level statement into a set of *assembly language* instructions for that CPU
  - The assembler translates each assembly language instruction into a *machine language* instruction
- In a CPU with a hardwired controller:
  - Each machine language instruction is decoded and executed
- In a CPU with a microcoded controller:
  - Each machine language instruction is defined by a set of *microinstructions*
  - Each microinstruction is decoded and executed

Types of Control

- Hardwired control
  - The controller decodes the contents of IR and the ALU status, and uses that information to control:
    - register muxes, ALU operation, and shifter
    - register loads and bus access (not shown)
- Microprogrammed control
  - Each machine language instruction is implemented by a set of *microinstructions*
    - The control store holds the full set of microinstructions (the *microprogram*)
    - The µIR (MicroInstruction Register) holds the current microinstruction
    - The µPC (Micro Program Counter) holds the address (in the control store) of the next microinstruction to be executed
  - The µcontroller decodes the contents of IR and the ALU status, along with the contents of the µIR, and uses that info…

Implementation of Simple Machine With Microcoded Control

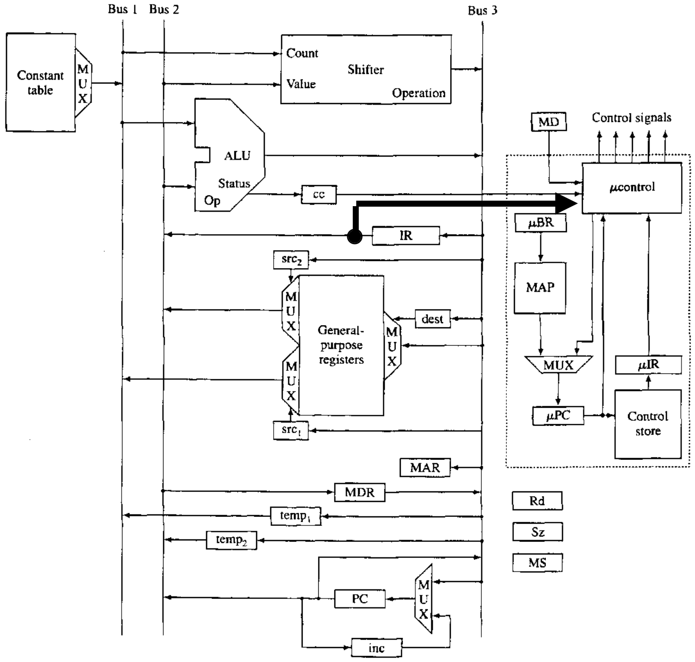

Diagram from *Computer Systems*, Maccabe, Irwin 1993

Microcode for “ADD” Instruction

```
ifetch:
    PC → MAR, READ word
    while !MD
    MDR → IR, inc PC

decode:
    SHIFT right (#11, IR) → µBR
    MAP µPC

add:
    ALU and (#15, IR) → src2
    SHIFT right (#4, IR) → temp2
    ALU and (#15, temp2) → src1
    SHIFT right (#4, temp2) → temp2
    ALU and (#15, temp2) → dest
    ALU add.cc (Reg[src1], Reg[src2]) → Reg[dest]
    branch ifetch
```

Writing a Microprogram in Microassembly Language

- A microprogram is written in a microassembly language, and stored in the control store (a ROM or PROM)
- Each microinstruction can (but does not have to) contain:
  - A label
    - Same as in assembly language programs
  - A control field
    - A “**while**” or “**if**” clause
  - An operation field
    - List of comma-separated *micro-operations*
  - A branch field
    - To implement branches

Micro-Operation Grammar

<micro op> :: <memory op> | <reg op> | inc PC | MAP µBR

<mem op> :: READ word | READ byte | WRITE word | WRITE byte

<reg op> :: <bus 3 source> → <bus 3 dest> | <bus 2 source> → MDR

<bus 3 source> :: <shift res> | <alu res> | MDR | PC

<bus 3 dest> :: µBR | IR | src2 | dest | Reg[dest] | src1 | MAR | temp1 | temp2 | PC

<shift res> :: SHIFT (shift op) (<bus 1 source>, <bus 2 source>)

<shift op> :: left | right | right arith

<alu res> :: ALU <alu op> (<bus 1 source>, <bus 2 source>)

<alu op> :: add | sub | and | or | xor | xorn | add.cc | sub.cc | and.cc | or.cc | xor.cc | xorn.cc

<bus 1 source> :: <constant> | Reg[src1] | temp1

<bus 2 source> :: IR | Reg[src2] | temp2 | PC

<constant> :: #n

Hardwired vs. Microprogrammed Control

- Hardwired control
  - Simple to implement
  - Fast (no extra level of instruction fetching, decoding, etc.)
- Microprogrammed control
  - Flexible — easier for designer to modify (microcode is stored in ROM, which can be changed fairly easily)
    - Microcode is classified as *firmware* — in between software and hardware
  - Allows convenient hardware / software tradeoffs — what the hardware doesn’t do (e.g., multiplication), do in microcode!
    - Supports families of machines with different price / performance tradeoffs
  - Provides support for very complex instructions