Microprogramming

Microprogramming Main Points/Terminology

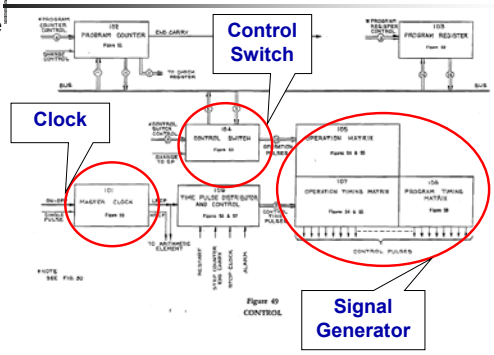
- Difference between **hardwired** control unit and **microprogrammed** control unit
- **Microprogram**
  - **Microinstruction**
  - **Macroinstruction**
- **Control store** and **micro-branching**
- **Horizontal** and **vertical** microprogramming

Important Trends/Concepts that Led to Microprogramming

- 1940's – **Stored program computers**
  - Program instructions can be stored in memory along with data and can be manipulated like data
- 1947 – MIT Whirlwind
  - real-time flight simulator
  - **control store**
    - **two-dimensional lattice**
      - **rows – time sequence**
      - **columns – control signals**

MIT Whirlwind

MIT Whirlwind Control Store



Historical Background (cont'd)

- 1951 – Maurice Wilkes
  - Sequencing of control signals within the computer was similar to the sequencing actions required in a regular program.
  - A stored program to represent the sequences of control signals. Called it **microprogramming**
    - Divide machine instructions into subinstructions **(microinstructions)** that implement the instruction set of the machine
    - Full set of microinstructions made up the microprogram

1970s

- Complex instruction sets
  - Trend towards instruction sets very similar to high-level languages

Important Trends that Led to Microprogramming

- ***Control logic design***
  - Complex and error-prone
  - Need a simpler method of developing the control logic for a computer
  - Writing a program
    - Simpler than designing the logic
    - Easier to change
- Benefits microprogramming offers
  - More complex instructions can be implemented
  - More primitive than assembly
  - Reduces field changes to defects

Is Microprogramming Still Used Today?

- ROM's used to be faster than RAM, but not any longer
  - Control stores were implemented on ROM
- Instruction sets have become much simpler (than the 1970s)
  - Less complexity
- Computer-aided design tools have improved

Yes, but benefits need to be weighed against costs: **Complex ISAs such as IA-32 sometimes have more complex instructions implemented as microprograms.**

Pentium, Pentium 4, special-purpose processors, etc.

Microprogramming: Basic Idea

- Recall control sequence for 1-bus SRC

Step	Concrete RTN	Control Sequence
T0.	MA ← PC: C ← PC+4;	PC_{out}, MA_{in}, Inc4, C_{in}, Read
T1.	MD ← M[MA]: PC ← C;	C_{out}, PC_{in}, Wait
T2.	IR ← MD;	MD_{out}, IR_{in}
T3.	A ← R[rb];	Grb, R_{out}, A_{in}
T4.	C ← A + R[rc];	Grc, R_{out}, ADD, C_{in}
T5.	R[ra] ← C;	C_{out}, Gra, R_{in}, End

- Control unit job is to generate the sequence of control signals
- How about building a computer to do this?

Microprogramming Process

1. Develop microinstruction set
2. Write microprogram
3. Microassemble microprogram
4. Place the microassembled program (microcode) onto PLA or ROM

The Microcode Engine

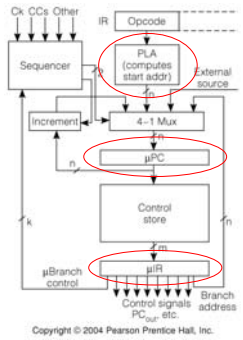
- A computer to generate control signals is much simpler than an ordinary computer
  - At the simplest, it just reads the control signals in order from a read only memory
- The memory is called the **control store**
  - **Separate from program memory**
- A control store word, or **microinstruction**, contains a bit pattern telling which control signals are true in a specific step
- The major issue is **sequencing**
  - **What order to read instructions?**

What Goes in a Microinstruction? Designing the Microinstruction set

- How many fields?
  - What does each field represent?
- What control signals are controlled by each field?
- How will sequencing (next microinstruction to execute) be determined?

Fig 5.22 Block Diagram of a Microcoded Control Unit

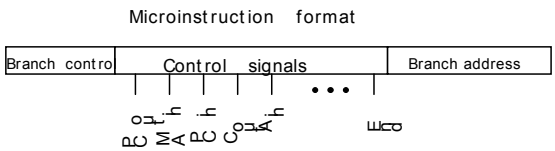
- **Microinstruction** has branch control, branch address, and control signal fields
- **Micro-program counter** can be set from several sources to do the required sequencing
  1. Next sequential
  2. Start address of next macro inst
  3. Microinst specified address



Parts of the Microprogrammed Control Unit

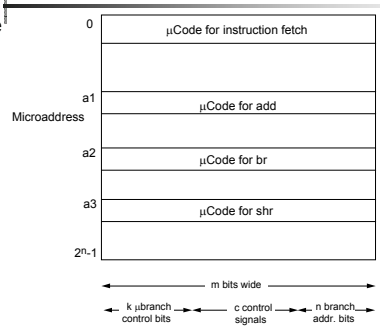
- Control signals are just read from memory, the main function is **sequencing**
- This is reflected in the several ways the μPC can be loaded
  1. Output of incrementer—μPC+1
  2. PLA output—start address for a macroinstruction
  3. Branch address from μinstruction
  4. External source—say for exception or reset
- Micro conditional branches can depend on condition codes, data path state, external signals, etc.

Contents of a Microinstruction



- Main component is list of 1/0 control signal values
- There is a branch address in the control store
- There are branch control bits to determine when to use the branch address and when to use μPC+1

Figure 5.23: Layout of the Control Store



- Common inst. fetch sequence
- Separate sequences for each (macro) instruction
- Wide words

Size and Shape of System RAM vs Control Store

- System RAM is one byte wide x 2^{32} bytes deep.
- Assume control store has 128 instructions, 128 bits wide, with 8 steps each.
- Control store would be 16 bytes wide, but only 128x8 or 1024 words deep.

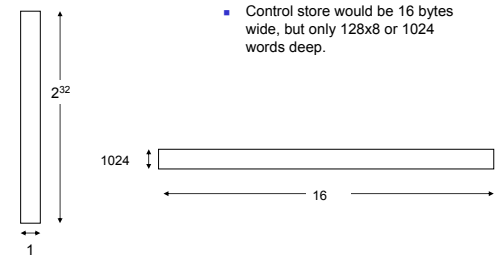


Table 5.2: Microinstruction Control Signals for the add Instruction

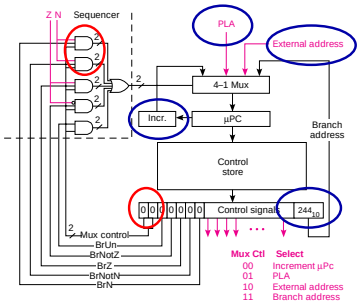
Address	Branch Control	PC_{out}	C_{out}	MD_{out}	R_{out}	MA_{in}	C_{in}	PC_{in}	IR_{in}	A_{in}	R_{in}	Inc4	Read	Wait	ADD	Gra	Grb	Grc	End
101	•••	1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0
102	•••	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0
103	•••	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
200	•••	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0
201	•••	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	1	0
202	•••	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1

- Addresses 101–103 are the instruction fetch
- Addresses 200–202 do the add
- Change of μcontrol from 103 to 200 uses a kind of μbranch

Uses for μbranching in the Microprogrammed Control Unit

- 1) Branch to start of μcode for a specific inst.
- 2) Conditional control signals, e.g. CON → PC_{in}
- 3) Looping on conditions, e.g. n≠0 → ... Goto6
- Conditions will control μbranches instead of being ANDed with control signals
- Microbranches are frequent and control store addresses are short, so it is reasonable to have a μbranch address field in every μ instruction

Fig 5.24 Branching Controls in the Microcoded Control Unit



- 5 branch conditions
  - NotN
  - N
  - NotZ
  - Z
  - Uncondit.
- To 1 of 4 places
  - Next μinst.
  - PLA
  - Extern. addr.
  - Branch addr.

Some Possible μbranches Using the Illustrated Logic

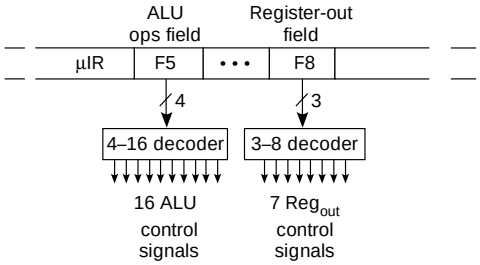
Mux Ctl	BrUn	BrNotZ	BrZ	BrNotN	BrN	Control Signals	Branch Address	Branching action
00	0	0	0	0	0	•••	XXX	None—next instruction
01	1	0	0	0	0	•••	XXX	Branch to output of PLA
10	0	0	1	0	0	•••	XXX	Br if Z to Extern. Addr.
11	0	0	0	0	1	•••	300	Br if N to 300 (else next)
11	0	0	0	1	0	0•••0	206	Br if $\bar{N}$ to 206 (else next)
11	1	0	0	0	0	•••	204	Br to 204

- If the control signals are all zero, the μinst. only does a test
- Otherwise test is combined with data path activity

Horizontal Versus Vertical Microcode Schemes

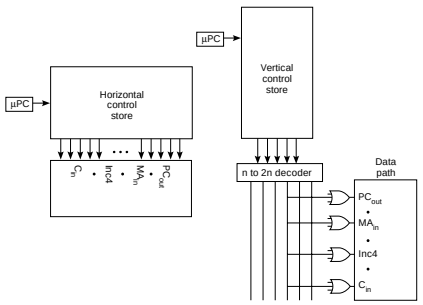
- In **horizontal** microcode, each control signal is represented by a bit in the μinstruction
  - Fewer control store words of more bits per word
- In **vertical** microcode, a set of true control signals is represented by a shorter code
  - Vertical μcode only allows RTs in a step for which there is a vertical μinstruction code
  - Thus vertical μcode may take more control store words of fewer bits

Fig 5.25 A Somewhat Vertical Encoding



- Scheme would save (16+7) - (4+3) = 16 bits/word in the case illustrated

Fig 5.26 Completely Horizontal and Vertical Microcoding



Saving Control Store Bits With Horizontal Microcode

- Some control signals cannot possibly be true at the same time
  - One and only one ALU function can be selected
  - Only one register out gate can be true with a single bus
  - Memory read and write cannot be true at the same step
- A set of m such signals can be encoded using $\log_2 m$ bits ($\log_2(m+1)$ to allow for no signal true)
- The raw control signals can then be generated by a k to 2^k decoder, where $2^k \geq m$ (or $2^k \geq m+1$)
- This is a compromise between horizontal and vertical encoding

A Microprogrammed Control Unit for the 1-bus SRC

- Using the 1-bus SRC data path design gives a specific set of control signals
- There are no condition codes, but data path signals CON and n=0 will need to be tested
- We will use μbranches BrCON, Brn=0, & Brn≠0
- We adopt the clocking logic of Fig. 4.9 on p. 4-20
- Logic for exception and reset signals is added to the microcode sequencer logic
- Exception and reset are assumed to have been synchronized to the clock

Table 5.4 Microinstructions for SRC add

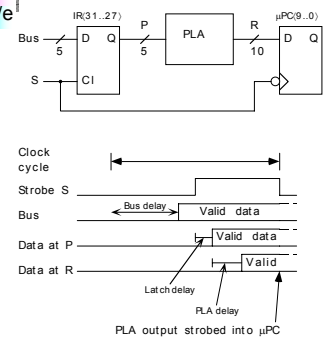
Addr.	Mux Ctl	BrUn	BrCON	Brn=0	Brn=0	End	PCout	MAin	Other Control Signals	Br Addr.	Actions
100	00	0	0	0	0	0	1	1	···	XXX	MA ← PC: C ← PC+4;
101	00	0	0	0	0	0	0	0	···	XXX	MD ← M[MA]: PC ← C;
102	01	1	0	0	0	0	0	0	···	XXX	IR ← MD; μPC ← PLA;
200	00	0	0	0	0	0	0	0	···	XXX	A ← R[rb];
201	00	0	0	0	0	0	0	0	···	XXX	C ← A + R[rc];
202	11	1	0	0	0	1	0	0	···	100	R[ra] ← C: μPC ← 100;

- Microbranching to the output of the PLA is shown at 102
- Microbranch to 100 at 202 starts next fetch

Getting the PLA Output in Time for the Microbranch

- So that the input to the PLA is correct for the μbranch in 102, it has to come from MD, not IR
- An alternative is to use see-thru latches for IR so the op code can pass through IR to PLA before the end of the clock cycle

See-thru Latch Hardware for IR So μPC Can Load Immediately



- Data must have time to get from MD across Bus, through IR, through the PLA, and satisfy μPC set up time before trailing edge of S

Fig 5.27 Microcode Sequencer Logic for SRC

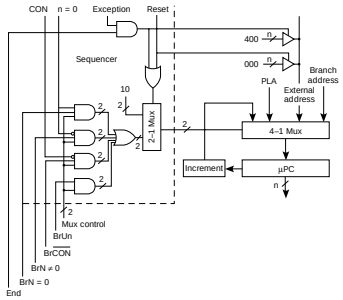


Table 5.6 A Somewhat Vertical Encoding of the SRC Microinstruction

F1	F2	F3	F4	F5	F6	F7	F8	F9
Mux Ctl	Branch control	End	Out signals	In signals	Misc.	Gate regs.	ALU	Branch address
00 01 10 11	000 BrUn 001 Br←CON 010 BrCON 011 Br n=0 100 Br n≠0 101 None	0 Cont. 1 End	000 PC_{out} 001 C_{out} 010 MD_{out} 011 R_{out} 100 BA_{out} 101 $c1_{out}$ 110 $c2_{out}$ 111 None	000 MA_{in} 001 PC_{in} 010 IR_{in} 011 A_{in} 100 R_{in} 101 MD_{in} 110 None	000 Read 001 Wait 010 Ld 011 Decr 100 CON_{in} 101 C_{in} 110 Stop 111 None	00 Gra 01 Grb 10 Grc 11 None	0000 ADD 0001 C=B 0010 SHR 0011 Inc4 · · · 1111 NOT	10 bits
2bits	3 bits	1 bit	3 bits	3 bits	3 bits	2 bits	4 bits	10 bits

Other Microprogramming Issues

- Multi-way branches: often an instruction can have 4-8 cases, say address modes
  - Could take 2-3 successive μbranches, i.e. clock pulses
  - The bits selecting the case can be ORed into the branch address of the μinstruction to get a several way branch
  - Say if 2 bits were ORed into the 3rd & 4th bits from the low end, 4 possible addresses ending in 0000, 0100, 1000, and 1100 would be generated as branch targets
  - Advantage is a multi-way branch in one clock
- A hardware push-down stack for the μPC can turn repeated μsequences into μsubroutines
- Vertical μcode can be implemented using a horizontal μengine, sometimes called nanocode