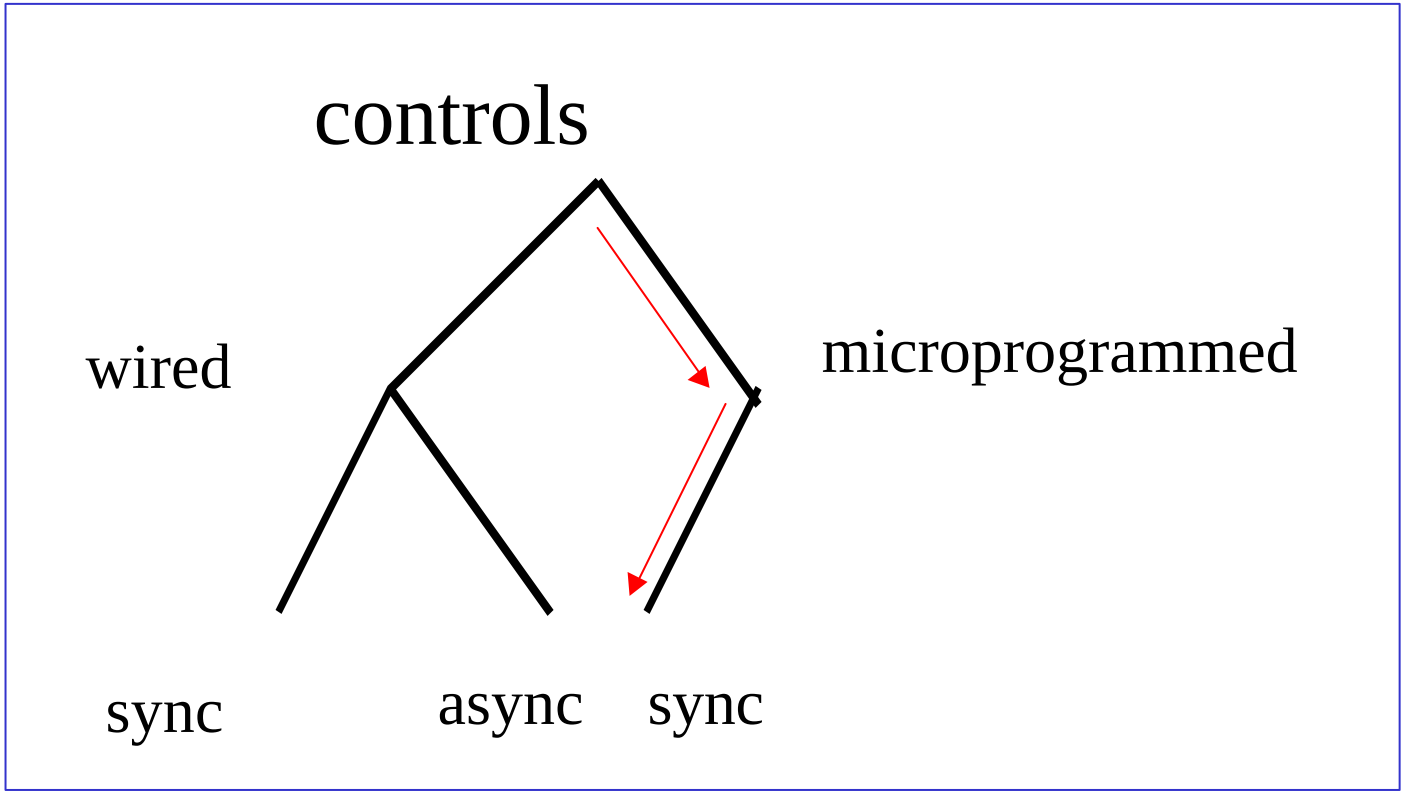
Part 9

Microprogramming:

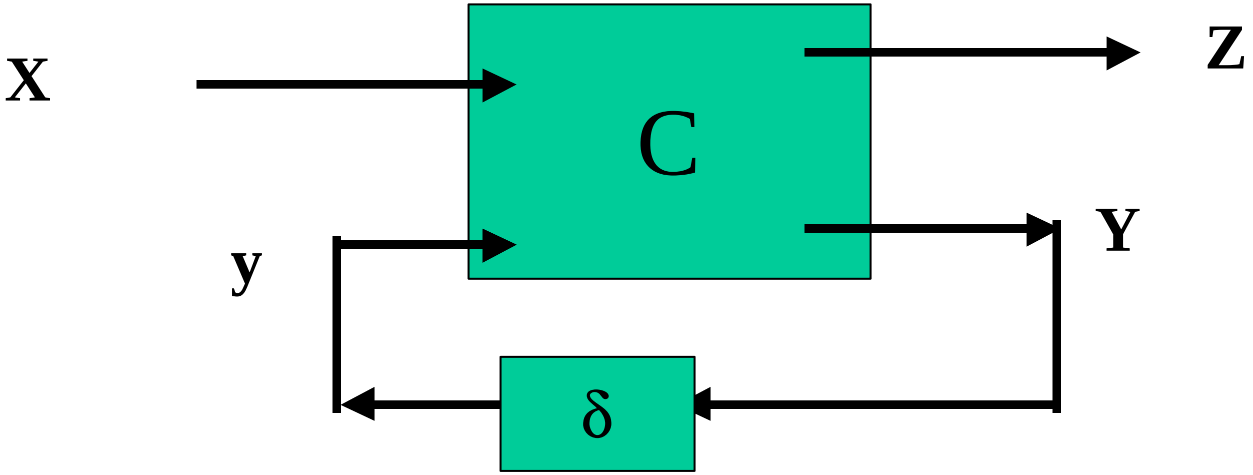
Implementing Instructions with Microsteps

We'll examine



- first, we need to study synchronous and asynchronous *models* of *sequential circuits*

Sequential Circuits



Where all boldface symbols are vector-valued

C = combinational logic

X = primary input vector   Z = primary output vector

y = feedback input vector   Y = feedback output vector

Sequential Circuits

Now

Z is not **f(X).** Rather

$\mathbf{Z} = \mathbf{Z}(\mathbf{X}; \mathbf{y})$

$\mathbf{Y} = \mathbf{Y}(\mathbf{X}; \mathbf{y})$ where

y is the ***system state***

Simple Sequential circuits:

- Flipflops:
  - SC
  - TRIGGER
    JK   etc etc

theorem

- Circuit s sequential => s has feedback loops
- converse is false

Back to Microprogramming (Tanenbaum’s Level 1)

- We’ll examine
  - synchronous, microprogrammed control
  - we have examined synchronous wired logic control (Simple Machine: Section 4 and MIPS chip: Section 5)
- Later we’ll examine
  - asynchronous wired logic control

Synchronous vs. Asynchronous: what’s the difference?

- Synchronous:
  - actions happen TIME = T
  - there is a *clock*
- Asynchronous:
  - action happens when previous action is complete -- no absolute time
  - no clock

Attributes

- Synchronous
  - much easier to design
  - perhaps 30% fewer gates
  - used in 100% of cpus today
  - changes in *gate delay* cause **trouble**

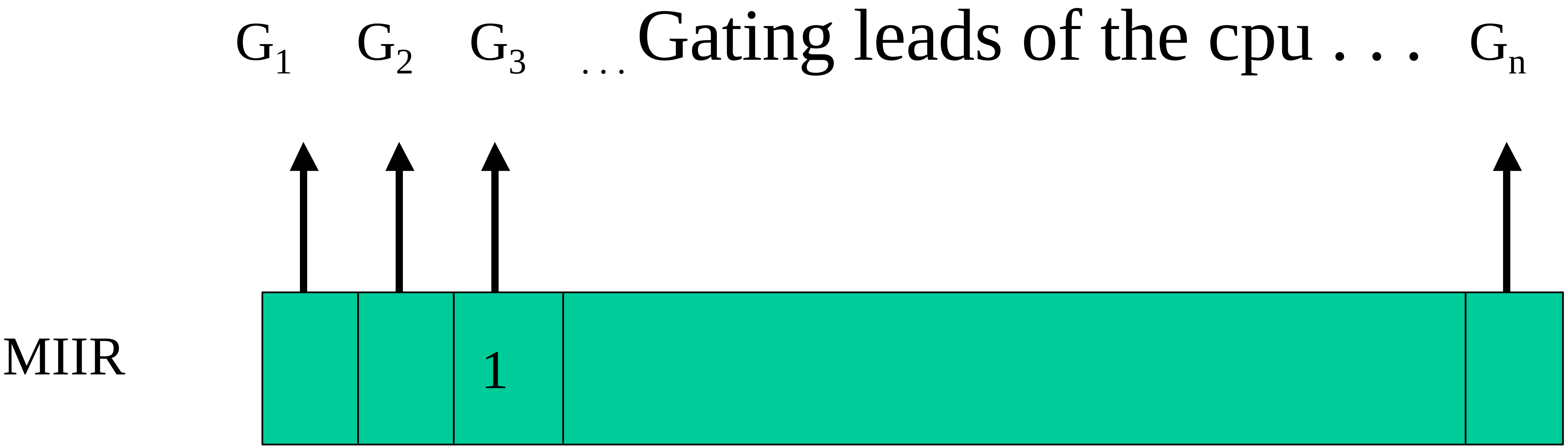
- Asynchronous
  - used between modules today (eg busses) but not in cpus

Synchronous Microprogrammed control

- The big idea:
  - connect all cpu gating leads to the bits of a register called MicroInstruction Instruction Register or MIIR
  - gate G_i is closed (active) iff

    c(MIIR[I]) = 1
  - there is one bit of MIIR per gating lead
  - sometimes called *horizontal* microprogramming

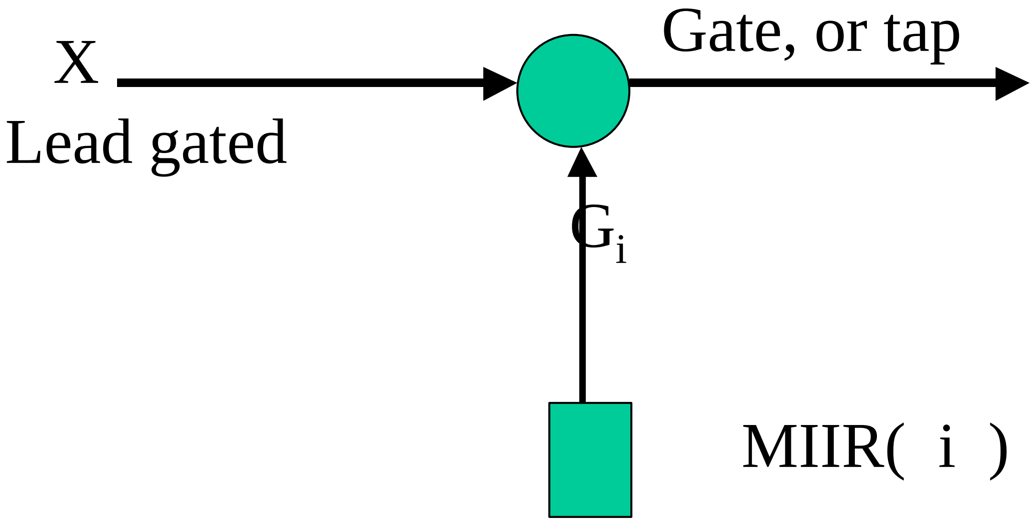
MIIR



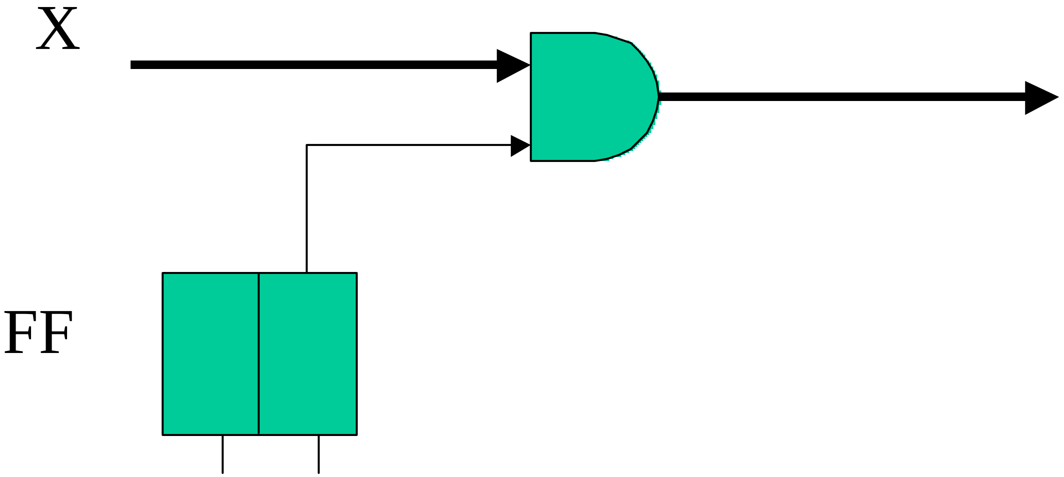
G_1 G_2 G_3 . . . Gating leads of the cpu . . . G_n

MIIR

Of course

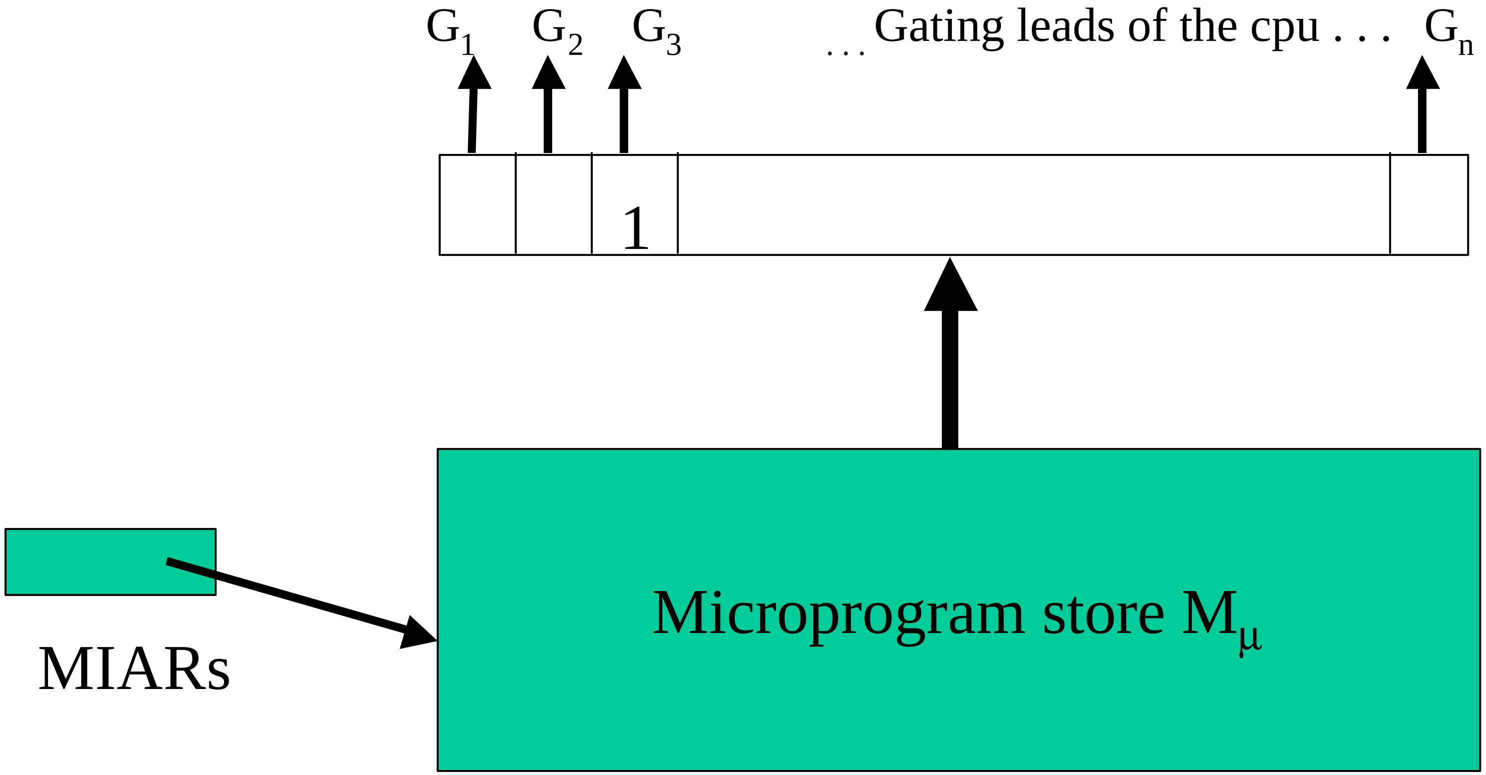


is actually



How to load words into the MIIR?

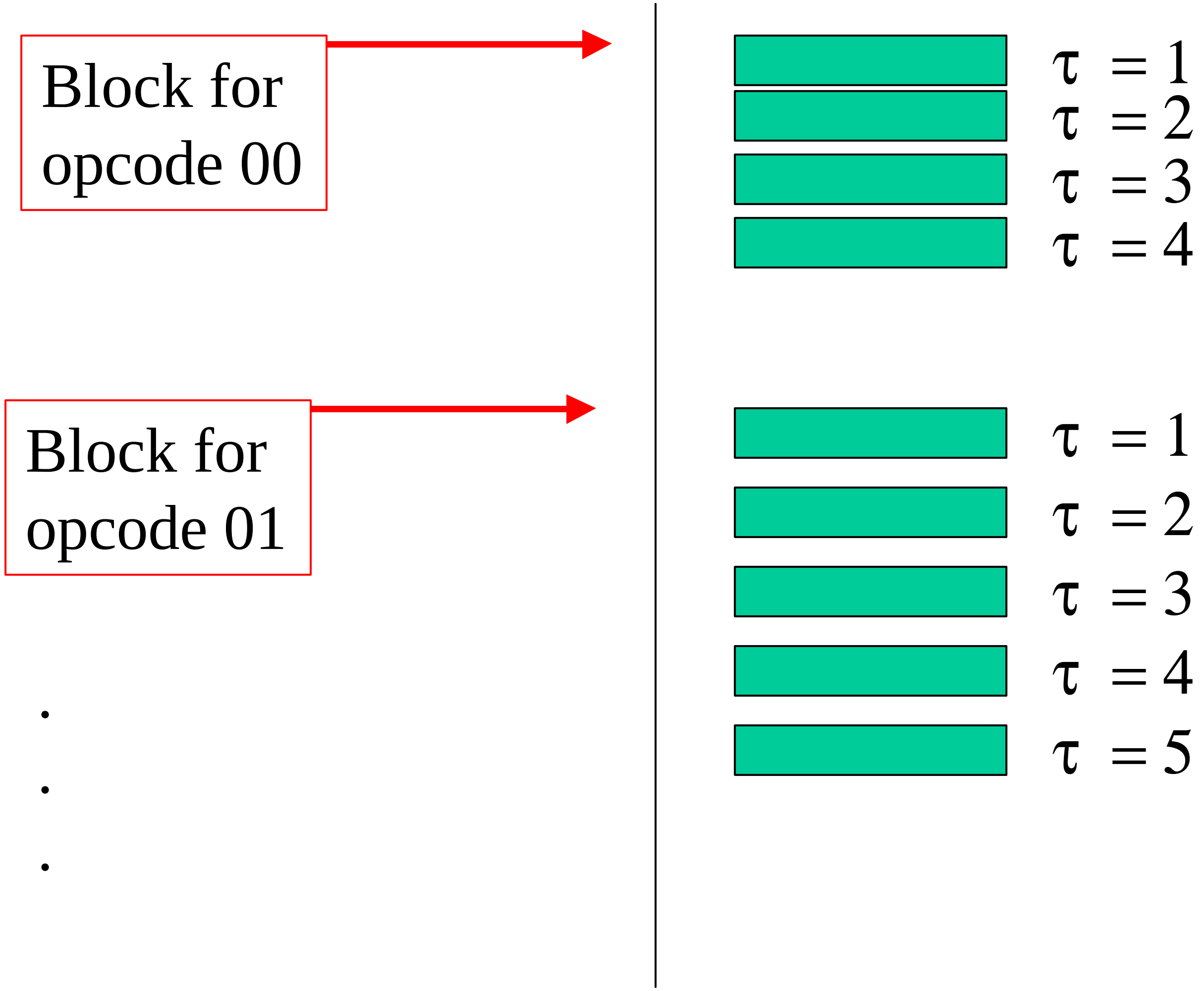
From a little store


G_1
G_2
G_3
. . . Gating leads of the cpu . . .
G_n
1
Microprogram store M_μ
MIARs

The Microinstruction Address Register MIAR

- Words are n bits wide = #(gating leads)
  - (maybe 500)
- if K = max(#(microsteps per instruction)) then
  - there is a block of K or fewer words in M_μ per instruction
  - opcode selects the first word
  - opcode + τ selects j^{th} word, $0 < j < K+1$ where τ is the value of the clock

How it looks:



Block for opcode 00

$\tau = 1$

$\tau = 2$

$\tau = 3$

$\tau = 4$

Block for opcode 01

$\tau = 1$

$\tau = 2$

$\tau = 3$

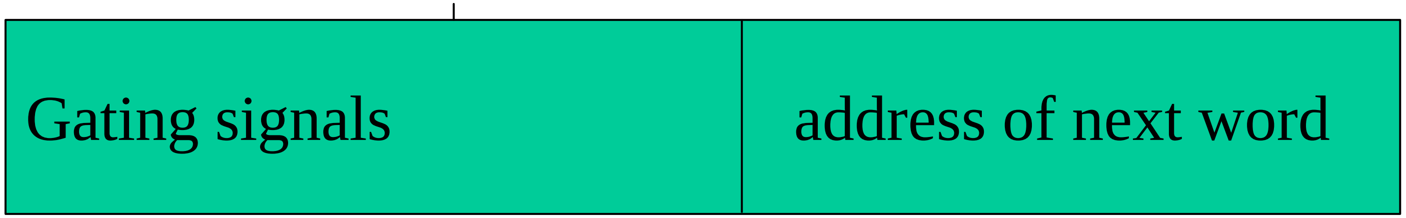
$\tau = 4$

$\tau = 5$

.
.
.

Embellishments:

- Jumps in microcode memory or store:
  - add an extra field to microprogram store words:



Gating signals

address of next word

Embellishments:

- Subroutines of microcode
  - FETCH, calculate effective address, etc

- constrained fields:

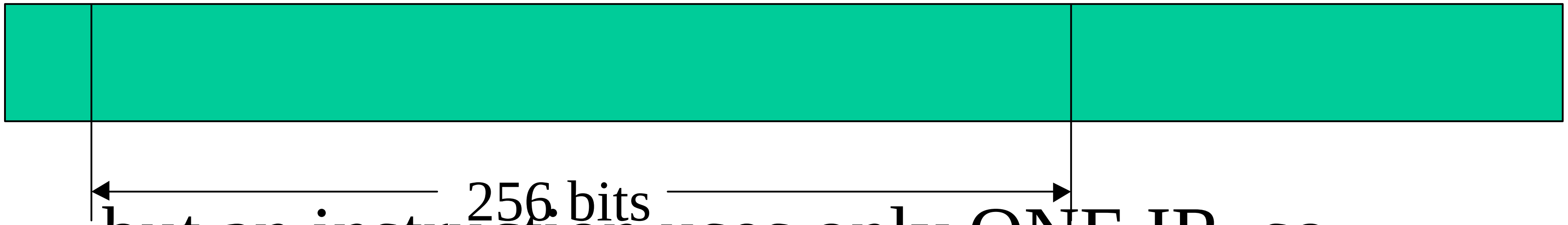
constrained fields:

- Suppose we have 256 Index Registers (IRs)

  hence 256 gating leads

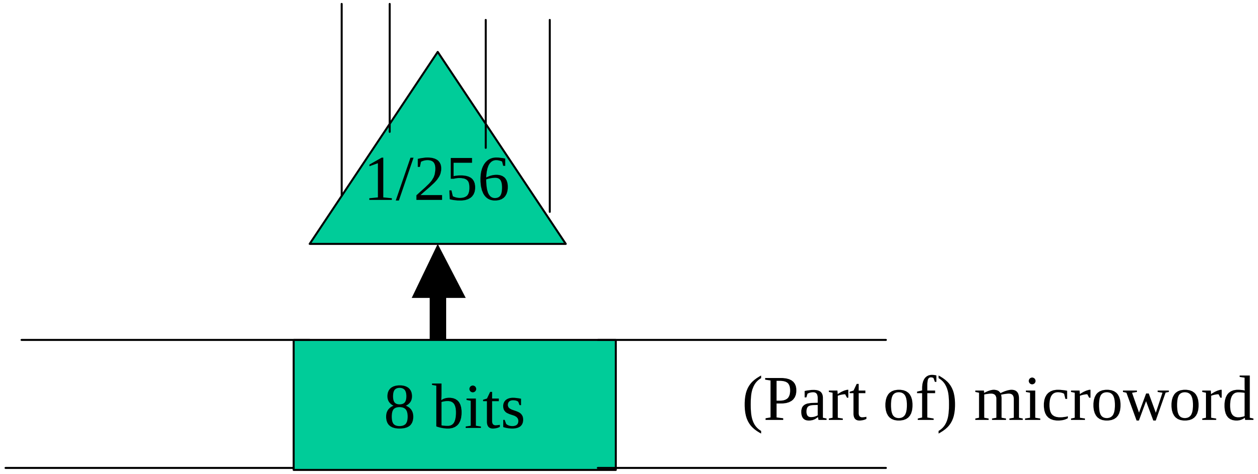
  in each Microword

  to select each of them



- but an instruction uses only ONE IR, so . . .

constrained fields:



- Pro: saved (256 - 8) bits of **each** microword
- Con: loss of future flexibillity.
  - – If you **ever** want an instruction using 2 IRs, . . .

constrained fields:

- Pushing this idea further we wind up with:

Micro opcode	IR	tag bits	memory address (Mp)

microword

- a little instruction! [called microinstruction]
- this style is called, *Vertical Microprogramming*

Retrospective

Effectively, we’ve built a cpu inside the cpu.

- Pro:
  - simpler hardware design, fewer parts
  - very flexible: rewire the cpu by changing microcode
- Con:
  - each instruction execution takes many microstore cycles - it must be FAST

Microprogram sequencers

- Definition: the thing which fetched microwords into the MIIR for execution

1] simplest:

```
Loop:   read microstore[MIAR]
        c[MMDR] --> MIIR
        WAIT                /* 1 minor clock cycle
        IF [tag_bit =1] GOTO Fetch_routine
        ELSE DO
                MIAR <-- c[MIAR] + 1
              OD
        GOTO Loop
```

Microprogram sequencers

2] Fancier:

multiple subroutines; subroutine linkage microinstruction

conditional transfers (for variable field-length operations etc)

Trends in microprogramming

- Pre-RISC:
  - user-alterable microprogram store
    - do-it-yourself instruction definition
  - universal use of it in microproessors
  - assemblers, register transfer languages, HLLS
- But RISC blew it all away (why??)
- used today in complex, slow I/O controllers

- Prof. Maurice V Wilkes’ idea,
  - see: “The Best Way to Design a Computer”

PH Break

- Now, read Section 5.5 of PH:
  - vertical microprogramming for the MIPS chip