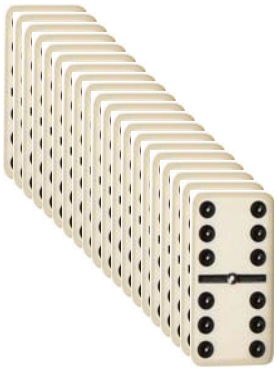


A
Unique
(*Singular*)
Architectural
Topic

What does
that mean?

In 2013, Stephen Dolan
Computer Lab University of Cambridge UK
wrote a paper showing …

Any x86 program can be written using only a *single* instruction

↓

mov

```
Start:  mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
end     Start
```

Every program looks like this

No … arithmetic, compares, jumps

Why consider this?

Removing all but *mov*

- **instruction format would be simplified**
- **expensive decode unit becomes cheaper**
- **silicon used for complex functional units could be repurposed as more cache**

However … it is a very long journey

Theory of only movs

Practice of only movs

Is this idea interesting?

Yes !

Is this idea practical?

Debatable !

Program size

Execution time

Code complexity

- Increase -

Christopher Domas
Cyber Security Researcher
Battelle Memorial Institute, Columbus OH

Accepted the challenge

Developed techniques for coding any high level language statement using only mov instructions

```
Start:  mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        end   Start
```

- Arithmetic & Logic
- Conditionals *If … Then*
- Jumps & Loops

3 types of instructions must be handled

- Arithmetic & Logic
  - Can be done using lookup tables

  Straightforward
- Conditionals *If … Then*
  - Without using conditional jumps (only have the mov instruction)
  - All instructions will be executed
  - How to avoid executing the *Then* when the *If* condition is not true

  Very clever
- Jumps & Loops

  Very clever & challenging

We present the idea in CSC236 as an

Architectural Knowledge Exercise

To keep the code manageable, the CSC236 implementation

- **data size is byte**
- **use int 21h to R/W data**
- **some programs have one jmp at the end of the code**

```
Start:  mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        int   21h   Read/Write data
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        mov
        jmp   Start
        end
```

Calculate z = x + y using byte variables

Arithmetic & Boolean Using lookup tables

Calculate z = x + y using byte variables

```
x       db   ?   ;variable x (0-127)
y       db   ?   ;variable y (0-127)
z       db   0   ;variable z (0-254)
```

Declare 3 byte size variables

X and y are limited so the sum x+y fits in a byte

Use x+y as an index into a table to get z

Calculate z = x + y using byte variables

```
x       db   ?   ;variable x (0-127)
y       db   ?   ;variable y (0-127)
z       db   0   ;variable z (0-254)

addtbl  db   000,001,002,003,004,005,006,007,008,009   ; Table
        db   010,011,012,013,014,015,016,017,018,019   ;   with
        ...                                            ;     values
        db   250,251,252,253,254,255                   ;       0-255
```

Declare a table with the values 0 to 255

representing the sum of x + y

Calculate z = x + y using byte variables

```
x       db   2   ;variable x (0-127)
y       db   4   ;variable y (0-127)
z       db   0   ;variable z (0-254)

addtbl  db   000,001,002,003,004,005,006,007,008,009   ; Table
        db   010,011,012,013,014,015,016,017,018,019   ;   with
        ...                                            ;     values
        db   250,251,252,253,254,255                   ;       0-255
```

Declare a table with the values 0 to 255

representing the sum of x + y

Use x + y as a double index to get z

Calculate z = x + y using byte variables

```
x       db   ?   ;variable x (0-127)
y       db   ?   ;variable y (0-127)
z       db   0   ;variable z (0-254)

addtbl  db   000,001,002,003,004,005,006,007,008,009   ; Table
        db   010,011,012,013,014,015,016,017,018,019   ;   with
        ...                                            ;     values
        db   250,251,252,253,254,255                   ;       0-255

        mov  bx,0                  ; clear the bx pointer
```

Bx is the only pointer register

that allows byte operations on bl

Calculate z = x + y using byte variables

```
x       db   ?   ;variable x (0-127)
y       db   ?   ;variable y (0-127)
z       db   0   ;variable z (0-254)

addtbl  db   000,001,002,003,004,005,006,007,008,009   ; Table
        db   010,011,012,013,014,015,016,017,018,019   ;   with
        ...                                            ;     values
        db   250,251,252,253,254,255                   ;       0-255

        mov  bx,0                  ; clear the bx pointer
        mov  bl,[x]                ; bx contains the value of x
        mov  si,bx                 ; si contains x for double indexing
```

Calculate z = x + y using byte variables

```
x       db   ?   ;variable x (0-127)
y       db   ?   ;variable y (0-127)
z       db   0   ;variable z (0-254)

addtbl  db   000,001,002,003,004,005,006,007,008,009   ; Table
        db   010,011,012,013,014,015,016,017,018,019   ;   with
        ...                                            ;     values
        db   250,251,252,253,254,255                   ;       0-255

        mov  bx,0                  ; clear the bx pointer
        mov  bl,[x]                ; bx contains the value of x
        mov  si,bx                 ; si contains x for double indexing
        mov  bl,[y]                ; bx contains the value of y
```

Calculate z = x + y using byte variables

```
x       db   ?   ;variable x  (0-127)
y       db   ?   ;variable y  (0-127)
z       db   0   ;variable z  (0-254)

addtbl  db   000,001,002,003,004,005,006,007,008,009    ; Table
        db   010,011,012,013,014,015,016,017,018,019    ;   with
        ...                                             ;   values
        db   250,251,252,253,254,255                    ;   0-255

        mov  bx,0               ; clear the bx pointer
        mov  bl,[x]             ; bx contains the value of x
        mov  si,bx              ; si contains x for double indexing
        mov  bl,[y]             ; bx contains the value of y
```

si has the value of x bx has the value of y

Calculate z = x + y using byte variables

```
x       db   ?   ;variable x  (0-127)
y       db   ?   ;variable y  (0-127)
z       db   0   ;variable z  (0-254)

addtbl  db   000,001,002,003,004,005,006,007,008,009    ; Table
        db   010,011,012,013,014,015,016,017,018,019    ;   with
        ...                                             ;   values
        db   250,251,252,253,254,255                    ;   0-255

        mov  bx,0               ; clear the bx pointer
        mov  bl,[x]             ; bx contains the value of x
        mov  si,bx              ; si contains x for double indexing
        mov  bl,[y]             ; bx contains the value of y
        mov  al,[addtbl + si + bx]   ; get value in table equal to x + y
```

Use the sum of x+y as an index into the table

Calculate z = x + y using byte variables

```
x       db   2   ;variable x  (0-127)
y       db   4   ;variable y  (0-127)
z       db   0   ;variable z  (0-254)

addtbl  db   000,001,002,003,004,005,006,007,008,009    ; Table
        db   010,011,012,013,014,015,016,017,018,019    ;   with
        ...                                             ;   values
        db   250,251,252,253,254,255                    ;   0-255

        mov  bx,0               ; clear the bx pointer
        mov  bl,[x]             ; bx contains the value of x
        mov  si,bx              ; si contains x for double indexing
        mov  bl,[y]             ; bx contains the value of y
        mov  al,[addtbl + 2 + 4]     ; get value in table equal to x + y
```

Use the sum of 2+4 as an index into the table

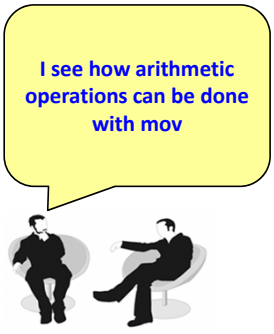
Calculate z = x + y using byte variables

```
x       db   ?   ;variable x  (0-127)
y       db   ?   ;variable y  (0-127)
z       db   0   ;variable z  (0-254)

addtbl  db   000,001,002,003,004,005,006,007,008,009    ; Table
        db   010,011,012,013,014,015,016,017,018,019    ;   with
        ...                                             ;   values
        db   250,251,252,253,254,255                    ;   0-255

        mov  bx,0               ; clear the bx pointer
        mov  bl,[x]             ; bx contains the value of x
        mov  si,bx              ; si contains x for double indexing
        mov  bl,[y]             ; bx contains the value of y
        mov  al,[addtbl + si + bx]   ; get value in table equal to x + y
        mov  [z],al             ; set z = x + y
```

Calculate z = x + y using byte variables

```
x       db   ?   ;variable x  (0-127)
y       db   ?   ;variable y  (0-127)
z       db   0   ;variable z  (0-254)

addtbl  db   000,001,002,003,004,005,006,007,008,009    ; Table
        db   010,011,012,013,014,015,016,017,018,019    ;   with
        ...                                             ;   values
        db   250,251,252,253,254,255                    ;   0-255

        mov  bx,0               ; clear the bx pointer
        mov  bl,[x]             ; bx contains the value of x
        mov  si,bx              ; si contains x for double indexing
        mov  bl,[y]             ; bx contains the value of y
        mov  al,[addtbl + si + bx]   ; get value in table equal to x + y
        mov  [z],al             ; set z = x + y
```

z=x+y using mov only



But the code for *if … then* requires a compare and jump

How can that be done with only mov?

```
if (x == y) then z = 'Y'

        mov   al,[x]
        cmp   al,[y]
        jne   endif
        mov   [z],'Y'
endif:
```

Key Concept

- One single code path
- All instructions execute no matter what
- *Force the code path to work on dummy data if we do not want to save the result*
- Use a selector to chose real or dummy data

Selector

Real data *If true*

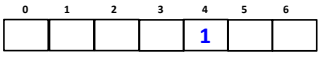
Dummy data *If false*

Execute the instructions, but not on the real data

Key Concept

Implementation

```
x=4  y=2     If (x == y)
mov [4], 1
```

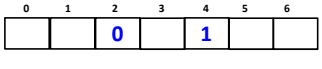


Work Memory

Use the Extra Segment for work memory

Move 1 to the mem location at offset x

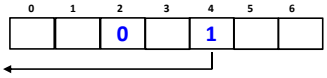
```
x=4  y=2     If (x == y)
mov [4], 1
mov [2], 0
```



Work Memory

Move 0 to the mem location at offset y

```
x=4  y=2     If (x == y)
mov [4], 1
mov [2], 0
```



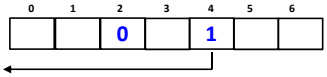
Work Memory

```
mov sel,[4]
```

Load selector from mem loc at offset x

What has happened ?

x=4 y=2 If (x == y)
mov [4], 1
mov [2], 0

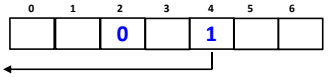


Work Memory

mov sel,[4]

x != y selector will be 1

x=4 y=2 If (x == y)
mov [4], 1
mov [2], 0

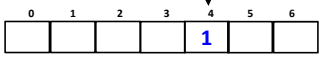


Work Memory

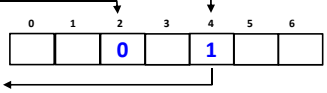
mov sel,[4]

x != y selector will be 1

x=4 y=4
mov [4], 1



x=4 y=2 If (x == y)
mov [4], 1
mov [2], 0

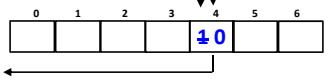


Work Memory

mov sel,[4]

x != y selector will be 1

x=4 y=4
mov [4], 1
mov [4], 0



mov sel,[4]

x = y selector will be 0

If (x == y) then ...

x = y selector will be 0

x != y selector will be 1

Now ... use the selector to implement or skip the then ... clause (use real or dummy data)

```
x       db    ?      ;variable x
y       db    ?      ;variable y
z       db    'N'    ;variable z
```

If (x==y) then z='Y'

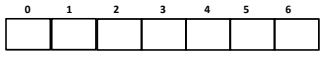
Declare x, y, z as byte variables (z initialized to 'N')

```
x       db    ?      ;variable x
y       db    ?      ;variable y
z       db    'N'    ;variable z
dum_z   db    ?      ;dummy variable z at memory offset z + 1
```

If (x==y) then z='Y'

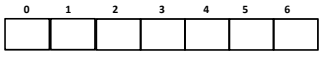
Declare dummy z right after z

```
x       db    ?       ;variable x
y       db    ?       ;variable y          If (x==y) then z='Y'
z       db    'N'     ;variable z
dum_z   db    ?       ;dummy variable z at memory offset z + 1

        .fardata
        db    256 dup(?)
```



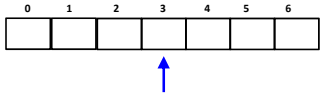
Declare 256 bytes of work memory
in the extra segment

```
x       db    ?       ;variable x
y       db    ?       ;variable y          If (x==y) then z='Y'
z       db    'N'     ;variable z
dum_z   db    ?       ;dummy variable z at memory offset z + 1

        .fardata
        db    256 dup(?)

mov   bx,0            ;bx pointer = 0000
```



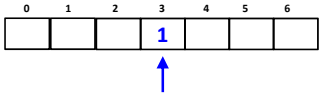
Clear the bx pointer register
(bx allows byte operations on bl)

```
x       db    3       ;variable x
y       db    ?       ;variable y          If (x==y) then z='Y'
z       db    'N'     ;variable z
dum_z   db    ?       ;dummy variable z at memory offset z + 1

        .fardata
        db    256 dup(?)

mov   bx,0            ;bx pointer = 0000
mov   bl,[x]          ;bx=x ... pts to mem addr=value_of_x
```



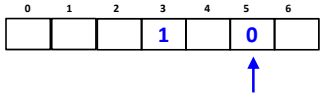
Move x to bl ... so bx points to the es memory
location equal to x ... for example x=3

```
x       db    3       ;variable x
y       db    ?       ;variable y          If (x==y) then z='Y'
z       db    'N'     ;variable z
dum_z   db    ?       ;dummy variable z at memory offset z + 1

        .fardata
        db    256 dup(?)

mov   bx,0            ;bx pointer = 0000
mov   bl,[x]          ;bx=x ... pts to mem addr=value_of_x
mov   byte ptr es:[bx],1  ;es:mem at addr=value_of_x set to 1
```



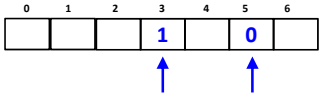
Move 1 to the es memory location
equal to x e.g. 3

```
x       db    3       ;variable x
y       db    5       ;variable y          If (x==y) then z='Y'
z       db    'N'     ;variable z
dum_z   db    ?       ;dummy variable z at memory offset z + 1

        .fardata
        db    256 dup(?)

mov   bx,0            ;bx pointer = 0000
mov   bl,[x]          ;bx=x ... pts to mem addr=value_of_x
mov   byte ptr es:[bx],1  ;es:mem at addr=value_of_x set to 1
mov   bl,[y]          ;bx=y ... pts to mem addr=value_of_y
mov   byte ptr es:[bx],0  ;es:mem at addr=value_of_y set to 0
```



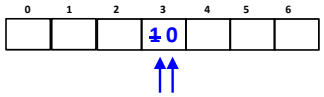
Move 0 to the es memory location
equal to y e.g. 5

```
x       db    3       ;variable x
y       db    5       ;variable y          If (x==y) then z='Y'
z       db    'N'     ;variable z
dum_z   db    ?       ;dummy variable z at memory offset z + 1

        .fardata
        db    256 dup(?)

mov   bx,0            ;bx pointer = 0000
mov   bl,[x]          ;bx=x ... pts to mem addr=value_of_x
mov   byte ptr es:[bx],1  ;es:mem at addr=value_of_x set to 1
mov   bl,[y]          ;bx=y ... pts to mem addr=value_of_y
mov   byte ptr es:[bx],0  ;es:mem at addr=value_of_y set to 0
```



Since x != y
these are different locations

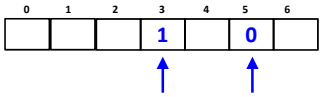
```
x       db    3       ;variable x
y       db    3       ;variable y          If (x==y) then z='Y'
z       db    'N'     ;variable z
dum_z   db    ?       ;dummy variable z at memory offset z + 1

        .fardata
        db    256 dup(?)
```




```
mov   bx,0              ;bx pointer = 0000

mov   bl,[x]            ;bx=x … pts to mem addr=value_of_x
mov   byte ptr es:[bx],1 ;es:mem at addr=value_of_x set to 1

mov   bl,[y]            ;bx=y … pts to mem addr=value_of_y
mov   byte ptr es:[bx],0 ;es:mem at addr=value_of_y set to 0
```

If x==y then that single es memory location would now be set to 0

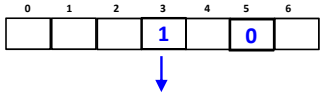
```
x       db    3       ;variable x
y       db    5       ;variable y          If (x==y) then z='Y'
z       db    'N'     ;variable z
dum_z   db    ?       ;dummy variable z at memory offset z + 1

        .fardata
        db    256 dup(?)
```




```
mov   bx,0              ;bx pointer = 0000

mov   bl,[x]            ;bx=x … pts to mem addr=value_of_x
mov   byte ptr es:[bx],1 ;es:mem at addr=value_of_x set to 1

mov   bl,[y]            ;bx=y … pts to mem addr=value_of_y
mov   byte ptr es:[bx],0 ;es:mem at addr=value_of_y set to 0
```

Since x != y
these are different locations

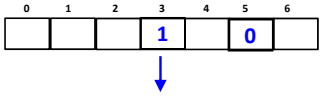
```
x       db    3       ;variable x
y       db    5       ;variable y          If (x==y) then z='Y'
z       db    'N'     ;variable z
dum_z   db    ?       ;dummy variable z at memory offset z + 1

        .fardata
        db    256 dup(?)
```




```
mov   bx,0              ;bx pointer = 0000

mov   bl,[x]            ;bx=x … pts to mem addr=value_of_x
mov   byte ptr es:[bx],1 ;es:mem at addr=value_of_x set to 1

mov   bl,[y]            ;bx=y … pts to mem addr=value_of_y
mov   byte ptr es:[bx],0 ;es:mem at addr=value_of_y set to 0

mov   bl,[x]            ;bx=x … pts to mem addr=value_of_x
mov   bl,es:[bx]        ;bx=0 if (x==y)   bx=1 if (x!=y)
```

Load selector (bx) from es mem loc equal to x

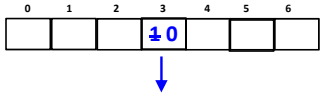
```
x       db    3       ;variable x
y       db    5       ;variable y          If (x==y) then z='Y'
z       db    'N'     ;variable z
dum_z   db    ?       ;dummy variable z at memory offset z + 1

        .fardata
        db    256 dup(?)
```




```
mov   bx,0              ;bx pointer = 0000

mov   bl,[x]            ;bx=x … pts to mem addr=value_of_x
mov   byte ptr es:[bx],1 ;es:mem at addr=value_of_x set to 1

mov   bl,[y]            ;bx=y … pts to mem addr=value_of_y
mov   byte ptr es:[bx],0 ;es:mem at addr=value_of_y set to 0

mov   bl,[x]            ;bx=x … pts to mem addr=value_of_x
mov   bl,es:[bx]        ;bx=0 if (x==y)   bx=1 if (x!=y)
```

bx=1 since x!=y

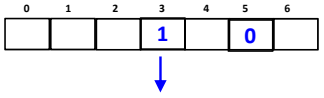
```
x       db    3       ;variable x
y       db    3       ;variable y          If (x==y) then z='Y'
z       db    'N'     ;variable z
dum_z   db    ?       ;dummy variable z at memory offset z + 1

        .fardata
        db    256 dup(?)
```




```
mov   bx,0              ;bx pointer = 0000

mov   bl,[x]            ;bx=x … pts to mem addr=value_of_x
mov   byte ptr es:[bx],1 ;es:mem at addr=value_of_x set to 1

mov   bl,[y]            ;bx=y … pts to mem addr=value_of_y
mov   byte ptr es:[bx],0 ;es:mem at addr=value_of_y set to 0

mov   bl,[x]            ;bx=x … pts to mem addr=value_of_x
mov   bl,es:[bx]        ;bx=0 if (x==y)   bx=1 if (x!=y)
```

bx would be 0 if x == y

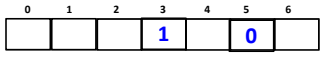
```
x       db    3       ;variable x
y       db    5       ;variable y          If (x==y) then z='Y'
z       db    'N'     ;variable z
dum_z   db    ?       ;dummy variable z at memory offset z + 1

        .fardata
        db    256 dup(?)
```




```
mov   bx,0              ;bx pointer = 0000

mov   bl,[x]            ;bx=x … pts to mem addr=value_of_x
mov   byte ptr es:[bx],1 ;es:mem at addr=value_of_x set to 1

mov   bl,[y]            ;bx=y … pts to mem addr=value_of_y
mov   byte ptr es:[bx],0 ;es:mem at addr=value_of_y set to 0

mov   bl,[x]            ;bx=x … pts to mem addr=value_of_x
mov   bl,es:[bx]        ;bx=0 if (x==y)   bx=1 if (x!=y)
```

bx is the selector 0 if (x==y) 1 if (x!=y)

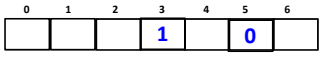
```
x       db    3       ;variable x
y       db    5       ;variable y          If (x==y) then z='Y'
z       db    'N'     ;variable z
dum_z   db    ?       ;dummy variable z at memory offset z + 1

        .fardata
        db    256 dup(?)
```




```
mov   bx,0              ;bx pointer = 0000

mov   bl,[x]            ;bx=x … pts to mem addr=value_of_x
mov   byte ptr es:[bx],1  ;es:mem at addr=value_of_x set to 1

mov   bl,[y]            ;bx=y … pts to mem addr=value_of_y
mov   byte ptr es:[bx],0  ;es:mem at addr=value_of_y set to 0

mov   bl,[x]             ;bx=x … pts to mem addr=value_of_x
mov   bl,es:[bx]         ;bx=0 if (x==y)    bx=1 if (x!=y)
```

Now use bx to select z or dum_z to get 'Y'

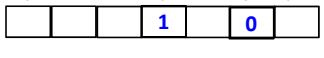
```
x       db    3       ;variable x
y       db    5       ;variable y          If (x==y) then z='Y'
z       db    'N'     ;variable z
dum_z   db    ?       ;dummy variable z at memory offset z + 1

        .fardata
        db    256 dup(?)
```




```
mov   bx,0              ;bx pointer = 0000

mov   bl,[x]            ;bx=x … pts to mem addr=value_of_x
mov   byte ptr es:[bx],1  ;es:mem at addr=value_of_x set to 1

mov   bl,[y]            ;bx=y … pts to mem addr=value_of_y
mov   byte ptr es:[bx],0  ;es:mem at addr=value_of_y set to 0

mov   bl,[x]             ;bx=x … pts to mem addr=value_of_x
mov   bl,es:[bx]         ;bx=0 if (x==y)    bx=1 if (x!=y)

mov   byte ptr[z+bx],'Y'
```

Mov 'Y' to the variable at location z + bx

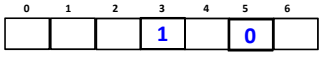
```
x       db    3       ;variable x
y       db    5       ;variable y          If (x==y) then z='Y'
z       db    'N'     ;variable z
dum_z   db    ?       ;dummy variable z at memory offset z + 1

        .fardata
        db    256 dup(?)
```




```
mov   bx,0              ;bx pointer = 0000

mov   bl,[x]            ;bx=x … pts to mem addr=value_of_x
mov   byte ptr es:[bx],1  ;es:mem at addr=value_of_x set to 1

mov   bl,[y]            ;bx=y … pts to mem addr=value_of_y
mov   byte ptr es:[bx],0  ;es:mem at addr=value_of_y set to 0

mov   bl,[x]             ;bx=x … pts to mem addr=value_of_x
mov   bl,es:[bx]         ;bx=0 if (x==y)    bx=1 if (x!=y)

mov   byte ptr[z+bx],'Y'
```

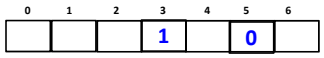
z+bx is z if bx=0 z+bx is dum_z if bx=1

```
x       db    3       ;variable x
y       db    5       ;variable y          If (x==y) then z='Y'
z       db    'N'     ;variable z
dum_z   db    ?       ;dummy variable z at memory offset z + 1

        .fardata
        db    256 dup(?)
```




```
mov   bx,0              ;bx pointer = 0000

mov   bl,[x]            ;bx=x … pts to mem addr=value_of_x
mov   byte ptr es:[bx],1  ;es:mem at addr=value_of_x set to 1

mov   bl,[y]            ;bx=y … pts to mem addr=value_of_y
mov   byte ptr es:[bx],0  ;es:mem at addr=value_of_y set to 0

mov   bl,[x]             ;bx=x … pts to mem addr=value_of_x
mov   bl,es:[bx]         ;bx=0 if (x==y)    bx=1 if (x!=y)

mov   byte ptr[z+bx],'Y'  ;z='Y' if (x==y)    dum_z='Y' if (x!=y)
```

```
x       db    3       ;variable x
y       db    5       ;variable y          If (x==y) then z='Y'
z       db    'N'     ;variable z
dum_z   db    ?       ;dummy variable z at memory offset z + 1

        .fardata
        db    256 dup(?)
```




```
mov   bx,0              ;bx pointer = 0000

mov   bl,[x]            ;bx=x … pts to mem addr=value_of_x
mov   byte ptr es:[bx],1  ;es:mem at addr=value_of_x set to 1

mov   bl,[y]            ;bx=y … pts to mem addr=value_of_y
mov   byte ptr es:[bx],0  ;es:mem at addr=value_of_y set to 0

mov   bl,[x]             ;bx=x … pts to mem addr=value_of_x
mov   bl,es:[bx]         ;bx=0 if (x==y)    bx=1 if (x!=y)

mov   byte ptr[z+bx],'Y'  ;z='Y' if (x==y)    dum_z='Y' if (x!=y)
```

if (x==y) then z='Y' with only mov

Looping and Branches

Looping and Branches

Have a flag that specifies whether *execution* is on or off

When you want to branch you turn execution off

For every operation perform these checks:

- If execution is on execute instruction on real data
- If execution is off
  - Is this instruction the branch target?
    - Yes – turn execution on & run on real data
    - No – leave execution off & run on dummy data

```
Start:
0000  mov …
0004  mov …
0008  mov …
000C  mov …
0010  mov …
0014  mov …
0018  mov …
001C  mov …
0020  mov …
0024  mov …
0028  jmp  Start
```

Only mov instructions

Except one unconditional jump from the end to the start*

*Christopher Domas shows how to eliminate that jmp

```
Start:
0000  mov …
0004  mov …
0008  mov …
000C  mov …
0010  mov …
0014  mov …
0018  mov …
001C  mov …
0020  mov …
0024  mov …
0028  jmp  Start
```

Branch Target

Execution: On

Only mov instructions

Except one unconditional jump from the end to the start*

*Christopher Domas shows how to eliminate that jmp

```
Start:
0000  mov …
0004  mov …
0008  mov …
000C  mov …
0010  mov …
0014  mov …
0018  mov …
001C  mov …   branch from here (001C)
0020  mov …
0024  mov …
0028  jmp  Start
```

Branch Target

Execution: On

```
Start:
0000  mov …
0004  mov …
0008  mov …   to here (0008)
000C  mov …
0010  mov …
0014  mov …
0018  mov …
001C  mov …   branch from here (001C)
0020  mov …
0024  mov …
0028  jmp  Start
```

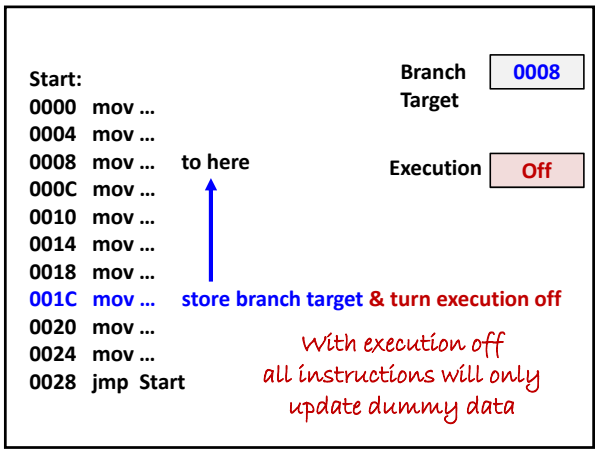
Branch Target

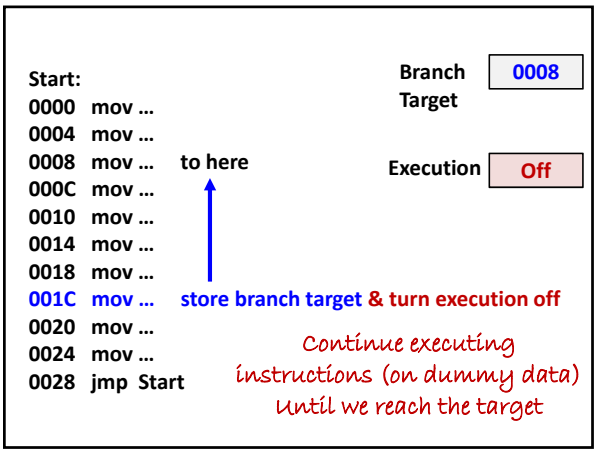
Execution: On

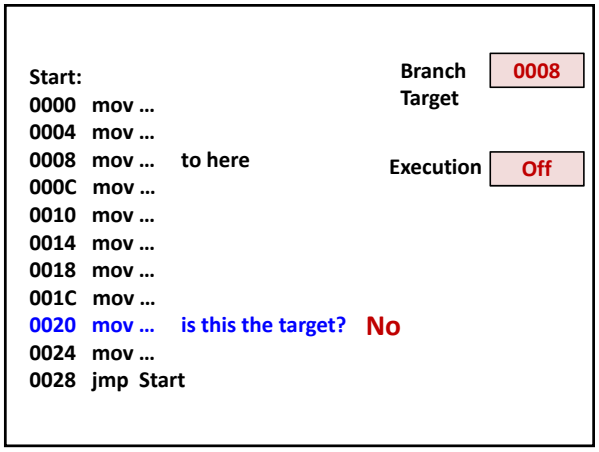
```
Start:
0000  mov …
0004  mov …
0008  mov …   to here
000C  mov …
0010  mov …
0014  mov …
0018  mov …
001C  mov …   store branch target
0020  mov …
0024  mov …
0028  jmp  Start
```

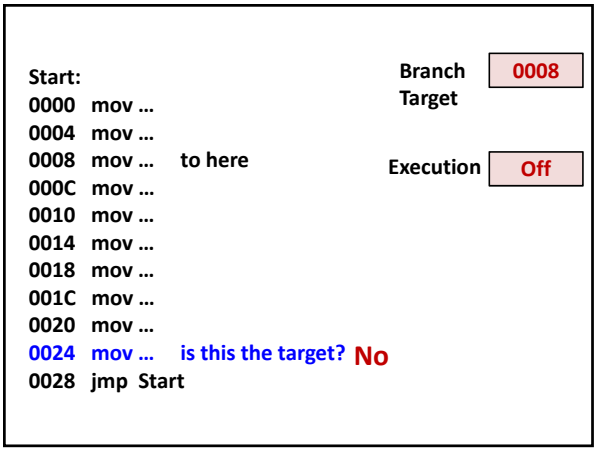
Branch Target: 0008

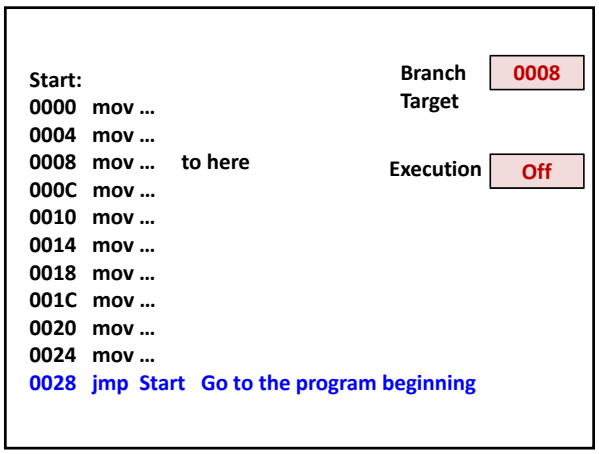
Execution: On

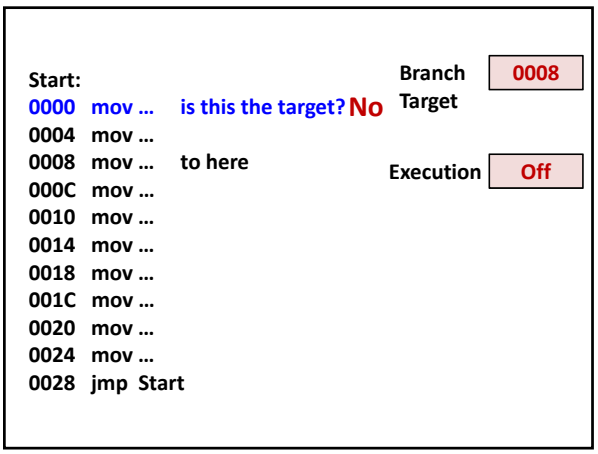

Start:
0000 mov …
0004 mov …
0008 mov … to here
000C mov …
0010 mov …
0014 mov …
0018 mov …
001C mov … store branch target & turn execution off
0020 mov …
0024 mov …
0028 jmp Start
Branch Target
0008
Execution
Off
With execution off all instructions will only update dummy data


Start:
0000 mov …
0004 mov …
0008 mov … to here
000C mov …
0010 mov …
0014 mov …
0018 mov …
001C mov … store branch target & turn execution off
0020 mov …
0024 mov …
0028 jmp Start
Branch Target
0008
Execution
Off
Continue executing instructions (on dummy data) until we reach the target


Start:
0000 mov …
0004 mov …
0008 mov … to here
000C mov …
0010 mov …
0014 mov …
0018 mov …
001C mov …
0020 mov … is this the target? No
0024 mov …
0028 jmp Start
Branch Target
0008
Execution
Off


Start:
0000 mov …
0004 mov …
0008 mov … to here
000C mov …
0010 mov …
0014 mov …
0018 mov …
001C mov …
0020 mov …
0024 mov … is this the target? No
0028 jmp Start
Branch Target
0008
Execution
Off


Start:
0000 mov …
0004 mov …
0008 mov … to here
000C mov …
0010 mov …
0014 mov …
0018 mov …
001C mov …
0020 mov …
0024 mov …
0028 jmp Start Go to the program beginning
Branch Target
0008
Execution
Off


Start:
0000 mov … is this the target? No
0004 mov …
0008 mov … to here
000C mov …
0010 mov …
0014 mov …
0018 mov …
001C mov …
0020 mov …
0024 mov …
0028 jmp Start
Branch Target
0008
Execution
Off

```
Start:
0000  mov ...
0004  mov ...    is this the target? No
0008  mov ...    to here
000C  mov ...
0010  mov ...
0014  mov ...
0018  mov ...
001C  mov ...
0020  mov ...
0024  mov ...
0028  jmp  Start
```

Branch Target: 0008

Execution: Off

```
Start:
0000  mov ...
0004  mov ...
0008  mov ...    is this the target?
000C  mov ...
0010  mov ...
0014  mov ...
0018  mov ...
001C  mov ...
0020  mov ...
0024  mov ...
0028  jmp  Start
```

Branch Target: 0008

Execution: Off

```
Start:
0000  mov ...
0004  mov ...
0008  mov ...    yes - target match
000C  mov ...
0010  mov ...
0014  mov ...
0018  mov ...
001C  mov ...
0020  mov ...
0024  mov ...
0028  jmp  Start
```

Branch Target: 0008

Execution: Off

```
Start:
0000  mov ...
0004  mov ...
0008  mov ...    yes - target match
000C  mov ...        - turn execution on
0010  mov ...
0014  mov ...
0018  mov ...
001C  mov ...
0020  mov ...
0024  mov ...
0028  jmp  Start
```

Branch Target: 0008

Execution: On

```
Start:
0000  mov ...
0004  mov ...
0008  mov ...    yes - target match
000C  mov ...        - turn execution on
0010  mov ...        - resume executing
0014  mov ...          instructions with real data
0018  mov ...
001C  mov ...
0020  mov ...
0024  mov ...
0028  jmp  Start
```

Branch Target: 0008

Execution: On

This testing expands and complicates the code ... but it does work

Sample Code Provided

Program: IfThen.asm

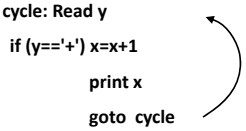
- Define 3 ASCII chars x, y z
- Read and echo x and y
- Only using mov, calculate:
  - z = 'N'
  - if (x == y) then z = 'Y'
- Output z
- Terminate

Straight Forward

Sample Code Provided

Program: JmpMov.asm

- Define 2 ASCII chars x, y
- Set x='0' … ASCII Char '0'
- cycle: Read y
  if (y=='+') x=x+1
  print x
  goto cycle
- Output y
- Terminate



Sample Code Provided

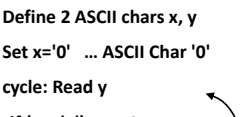
Program: JmpMov.asm

- Define 2 ASCII chars x, y
- Set x='0' … ASCII Char '0'
- cycle: Read y
  if (y=='+') x=x+1
  print x
  goto cycle
- Output y
- Terminate

Sample Code Provided

Program: JmpMov.asm

- Define 2 ASCII chars x, y
- Set x='0' … ASCII Char '0'
- cycle: Read y
  if (y=='+') x=x+1
  print x
  goto cycle
- Output y
- Terminate



Shows branch target testing

MOV assignment

Spec on the WEB - Optional team assignment

Program: mov.asm

- Define 3 ASCII chars x, y, z
- Read and echo x, y, z
  x will be an ASCII character 'A'-'Z' or '0'-'9'
  y will be an ASCII digit '0' - '9'
  z can be an ASCII '+' or other
- Using only the mov instruction
  - convert y from ASCII '0'-'9' to hex 0-9
  - if (z == '+') then x = x + y
- Output x and terminate

Use *IfThen* & *JmpMov* code as models

x	y	z	x
A	3	+	D
2	5	/	2