

CSCI 2330



TODO

- Assignment 2 assigned due October 7
- Study for Exam 1 October 9

Overview for today

- If
- Do
- While
- For

Condition codes



How do we implement branching?

- We first need to understand condition codes
- These are effectively special 1-bit registers
- They get automatically updated after most arithmetic instructions based on the result
- x86 processors have at least four...

Control codes

- What are they exactly?
- How are they assigned?
- How do we read them?
- How do we use them for control flow?

How are control codes assigned?

- Based on the value of the **destination register** after an arithmetic operation
- Exception: LEA does not update codes

[illegible]

Burying the lede

- Like, sure, you can say:
- `int foo = x < y;`
- But where do we *typically* see expressions like “`x < y`”?

Jumps

- We've seen a special form of them via `callq`
- In C, closest analog is `goto...`
- Syntax: `goto label_name;`

goto.c

```
#include <stdio.h>

int main() {
    int n = 0;

    // If the number is zero, skip to end
    if (n != 0) {
        // This will be skipped
        printf("You entered: %d\n", n);
    }

    printf("Exiting the program.\n");
    return 0;
}
```

goto.c

```
#include <stdio.h>

int main() {
    int n = 0;

    // If the number is zero, skip to end
    if (n == 0)
        goto jump_here;

    // This will be skipped
    printf("You entered: %d\n", n);

jump_here:
    printf("Exiting the program.\n");
    return 0;
}
```

In assembly...

- All we have are gotos...
- No loop constructs
- No if statements
- We must construct all of these using gotos!

jmp label_name

- The x86 assembly equivalent of a goto!
- Normally %rip becomes address of next instruction in memory (sequential execution)
- jmp label_name causes %rip to become address of label_name

jmp label_name

- This is called an *unconditional* jump because the jump is **always** taken
- But what if we want an if statement? Or a (non-infinite) loop?
- Then we need a *conditional* jump...

1	1.1.1.1	1.1.1.1
2	1.1.1.2	1.1.1.2
3	1.1.1.3	1.1.1.3
4	1.1.1.4	1.1.1.4
5	1.1.1.5	1.1.1.5
6	1.1.1.6	1.1.1.6
7	1.1.1.7	1.1.1.7
8	1.1.1.8	1.1.1.8
9	1.1.1.9	1.1.1.9
10	1.1.1.10	1.1.1.10
11	1.1.1.11	1.1.1.11
12	1.1.1.12	1.1.1.12
13	1.1.1.13	1.1.1.13
14	1.1.1.14	1.1.1.14
15	1.1.1.15	1.1.1.15
16	1.1.1.16	1.1.1.16
17	1.1.1.17	1.1.1.17
18	1.1.1.18	1.1.1.18
19	1.1.1.19	1.1.1.19
20	1.1.1.20	1.1.1.20
21	1.1.1.21	1.1.1.21
22	1.1.1.22	1.1.1.22
23	1.1.1.23	1.1.1.23
24	1.1.1.24	1.1.1.24
25	1.1.1.25	1.1.1.25
26	1.1.1.26	1.1.1.26
27	1.1.1.27	1.1.1.27
28	1.1.1.28	1.1.1.28
29	1.1.1.29	1.1.1.29
30	1.1.1.30	1.1.1.30
31	1.1.1.31	1.1.1.31
32	1.1.1.32	1.1.1.32
33	1.1.1.33	1.1.1.33
34	1.1.1.34	1.1.1.34
35	1.1.1.35	1.1.1.35
36	1.1.1.36	1.1.1.36
37	1.1.1.37	1.1.1.37
38	1.1.1.38	1.1.1.38
39	1.1.1.39	1.1.1.39
40	1.1.1.40	1.1.1.40
41	1.1.1.41	1.1.1.41
42	1.1.1.42	1.1.1.42
43	1.1.1.43	1.1.1.43
44	1.1.1.44	1.1.1.44
45	1.1.1.45	1.1.1.45
46	1.1.1.46	1.1.1.46
47	1.1.1.47	1.1.1.47
48	1.1.1.48	1.1.1.48
49	1.1.1.49	1.1.1.49
50	1.1.1.50	1.1.1.50
51	1.1.1.51	1.1.1.51
52	1.1.1.52	1.1.1.52
53	1.1.1.53	1.1.1.53
54	1.1.1.54	1.1.1.54
55	1.1.1.55	1.1.1.55
56	1.1.1.56	1.1.1.56
57	1.1.1.57	1.1.1.57
58	1.1.1.58	1.1.1.58
59	1.1.1.59	1.1.1.59
60	1.1.1.60	1.1.1.60
61	1.1.1.61	1.1.1.61
62	1.1.1.62	1.1.1.62
63	1.1.1.63	1.1.1.63
64	1.1.1.64	1.1.1.64
65	1.1.1.65	1.1.1.65
66	1.1.1.66	1.1.1.66
67	1.1.1.67	1.1.1.67
68	1.1.1.68	1.1.1.68
69	1.1.1.69	1.1.1.69
70	1.1.1.70	1.1.1.70
71	1.1.1.71	1.1.1.71
72	1.1.1.72	1.1.1.72
73	1.1.1.73	1.1.1.73
74	1.1.1.74	1.1.1.74
75	1.1.1.75	1.1.1.75
76	1.1.1.76	1.1.1.76
77	1.1.1.77	1.1.1.77
78	1.1.1.78	1.1.1.78
79	1.1.1.79	1.1.1.79
80	1.1.1.80	1.1.1.80
81	1.1.1.81	1.1.1.81
82	1.1.1.82	1.1.1.82
83	1.1.1.83	1.1.1.83
84	1.1.1.84	1.1.1.84
85	1.1.1.85	1.1.1.85
86	1.1.1.86	1.1.1.86
87	1.1.1.87	1.1.1.87
88	1.1.1.88	1.1.1.88
89	1.1.1.89	1.1.1.89
90	1.1.1.90	1.1.1.90
91	1.1.1.91	1.1.1.91
92	1.1.1.92	1.1.1.92
93	1.1.1.93	1.1.1.93
94	1.1.1.94	1.1.1.94
95	1.1.1.95	1.1.1.95
96	1.1.1.96	1.1.1

How do we do if statements in x86?

- I.e., conditional branching?
- x86 has no concept of “then” or “else” blocks
- But it has conditional jumps!



Notice: the comparison in `if (...)` was logically negated – why?



How do we do loops in x86?

- “Do” loops? “While” loops? “For” loops?
- We will start with do loops because they are the simplest to translate
- Then show while and for

Do loops

Do loops

While loops (jump to middle)

While loops (jump to middle)

- This is a straight translation of while loops into gotos

While loops (guarded do)



While loops (guarded do)

- We transform the while loop into a do loop by adding one additional check at the beginning
- Two approaches: what are the tradeoffs?



Guarded do optimization



For loops

- Can we rewrite a for loop as a while loop?



For loops

- Can we rewrite a for loop as a while loop? Yes.
- Guarded-Do optimization often applies!
- `for (i = 0; i < 100; ++i) { ... }`

```
    i = 0;
    if (i >= 100) goto done;
loop:
    ...
    ++i;
    if (i < 100) goto loop;
done:
```

For loops

- Can we rewrite a for loop as a while loop? Yes.
- Guarded-Do optimization often applies!
- `for (i = 0; i < 100; ++i) { ... }`

```
    i = 0;  
    if (i >= 100) goto done;  
loop:  
    ...  
    ++i;  
    if (i < 100) goto loop;  
done:
```

gccopt

- Let's see how a real-world compiler optimizes loops!

Reverse engineering exercises

TODO

- Assignment 2 assigned due October 7
- Study for Exam 1 October 9