COMPUTING MACHINERY

CPSC 355

RAHEELA AFZAL

raheela.afzal1@ucalgary.ca

OPERATIONS

INSTRUCTION	MEANING
MOV Xd, Xn	Xd = Xn
ADD Xd, Xn, Xm,	Xd = Xn + Xm
SUB Xd, Xn, Xm	Xd = Xn - Xm.
MADD Xd, Xn, Xm, Xa	Xd=(Xn × Xm) + Xa
MSUB Xd, Xn, Xm, Xa	Xd = Xa - (Xn × Xm.)

BRANCH

Unconditional Branch (immediate)

`B label`

Branch: unconditionally jumps to pc-relative label.

BL label

Branch and Link: unconditionally jumps to pc-relative label, writing the address of the next sequential instruction to register X30.

```
again: mov x19, 20
mov x20, #10
add x22, x19, x20
b again
```

Conditional Branch

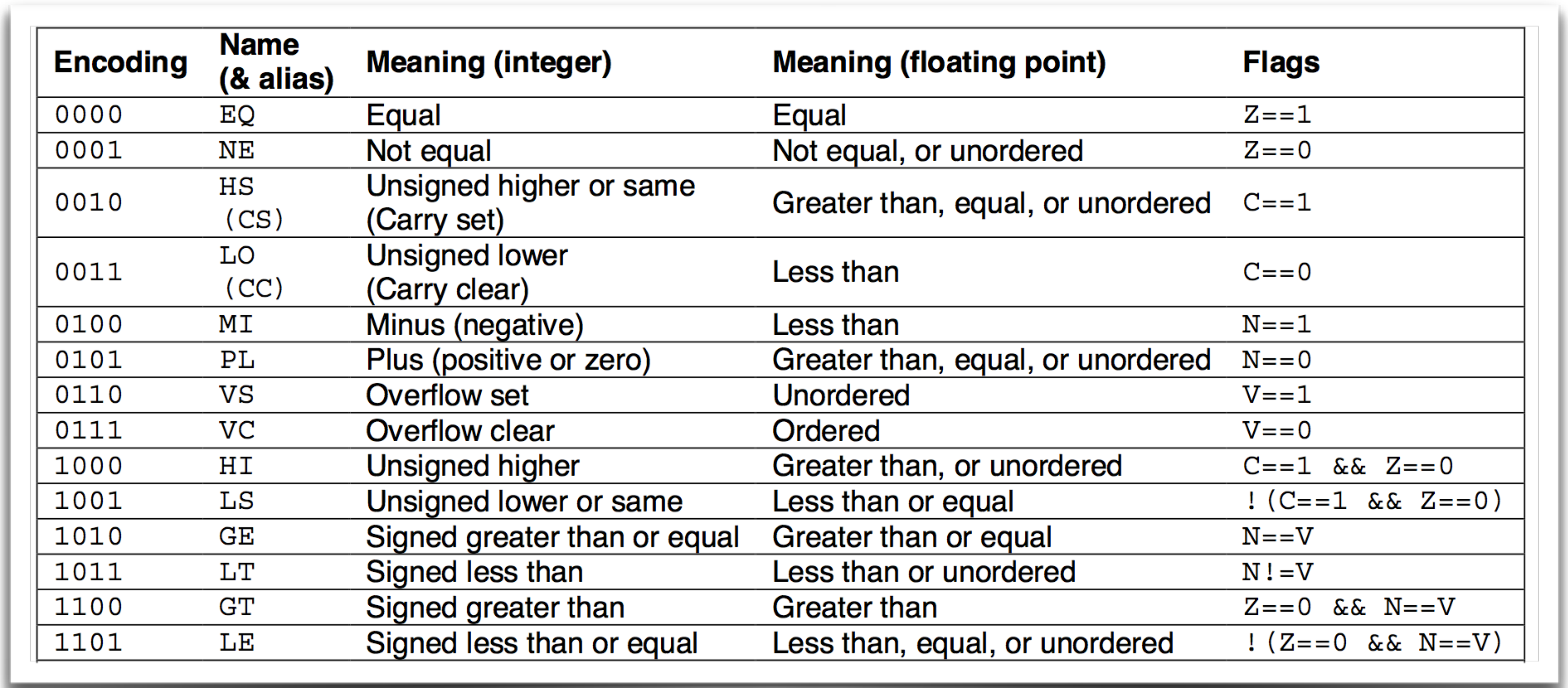
`B.cond label`

Branch: conditionally jumps to program-relative label if condition is true.

Refer to next slide for various condition codes.

```
mov x19, 1
mov x20, 20
label: add x19, x19, 1
cmp x19, x20
b.le label
```

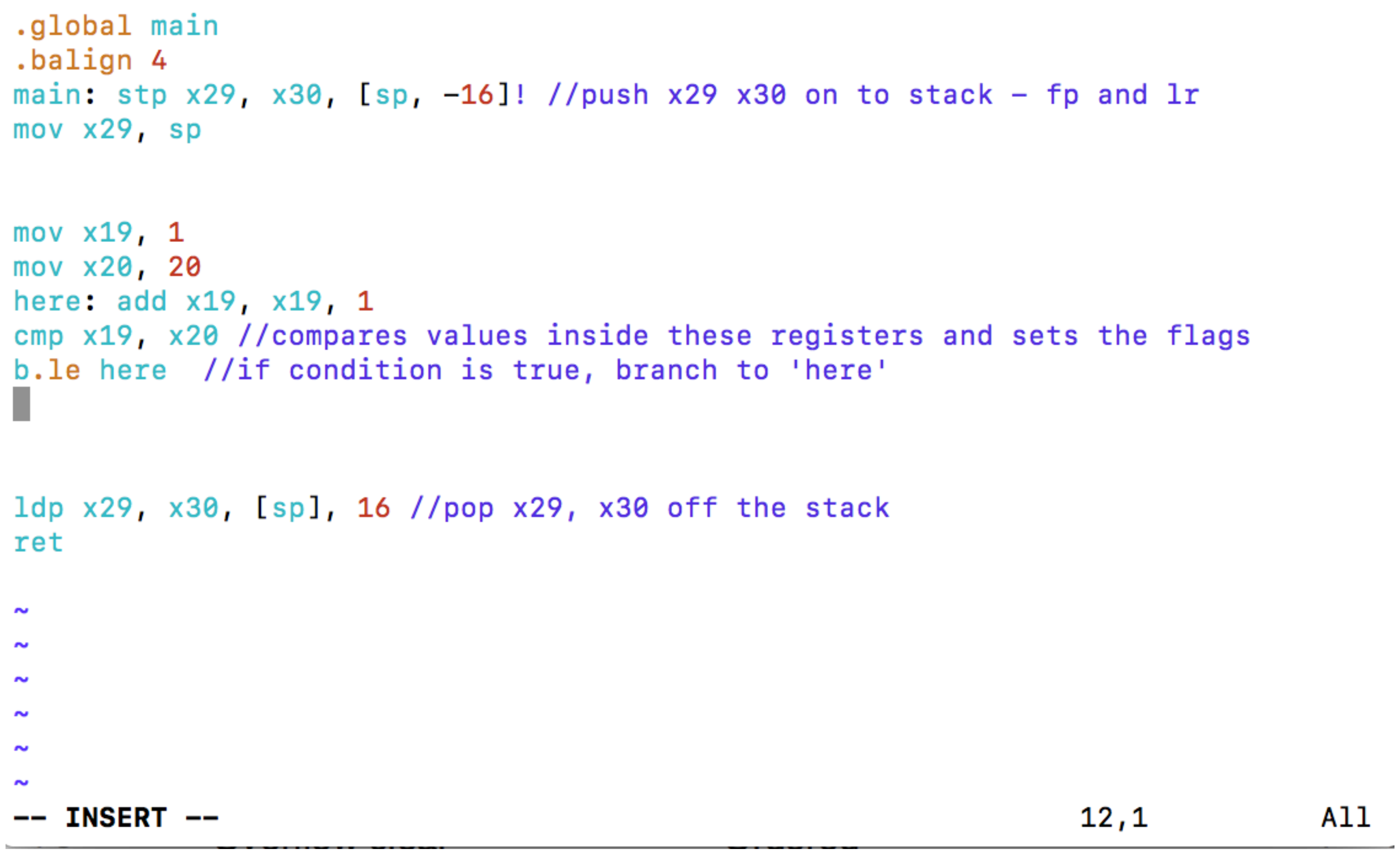
Condition Codes



Encoding	Name (& alias)	Meaning (integer)	Meaning (floating point)	Flags
`0000`	`EQ`	Equal	Equal	`Z==1`
`0001`	`NE`	Not equal	Not equal, or unordered	`Z==0`
`0010`	`HS` `(CS)`	Unsigned higher or same (Carry set)	Greater than, equal, or unordered	`C==1`
`0011`	`LO` `(CC)`	Unsigned lower (Carry clear)	Less than	`C==0`
`0100`	`MI`	Minus (negative)	Less than	`N==1`
`0101`	`PL`	Plus (positive or zero)	Greater than, equal, or unordered	`N==0`
`0110`	`VS`	Overflow set	Unordered	`V==1`
`0111`	`VC`	Overflow clear	Ordered	`V==0`
`1000`	`HI`	Unsigned higher	Greater than, or unordered	`C==1 && Z==0`
`1001`	`LS`	Unsigned lower or same	Less than or equal	`!(C==1 && Z==0)`
`1010`	`GE`	Signed greater than or equal	Greater than or equal	`N==V`
`1011`	`LT`	Signed less than	Less than or unordered	`N!=V`
`1100`	`GT`	Signed greater than	Greater than	`Z==0 && N==V`
`1101`	`LE`	Signed less than or equal	Less than, equal, or unordered	`!(Z==0 && N==V)`

SAMPLE ASSEMBLY LANGUAGE PROGRAM

Code: loopp.s



```
.global main
.balign 4
main: stp x29, x30, [sp, -16]! //push x29 x30 on to stack - fp and lr
mov x29, sp


mov x19, 1
mov x20, 20
here: add x19, x19, 1
cmp x19, x20 //compares values inside these registers and sets the flags
b.le here  //if condition is true, branch to 'here'



ldp x29, x30, [sp], 16 //pop x29, x30 off the stack
ret

~
~
~
~
~
~
-- INSERT --                                                   12,1          All
```

GDB

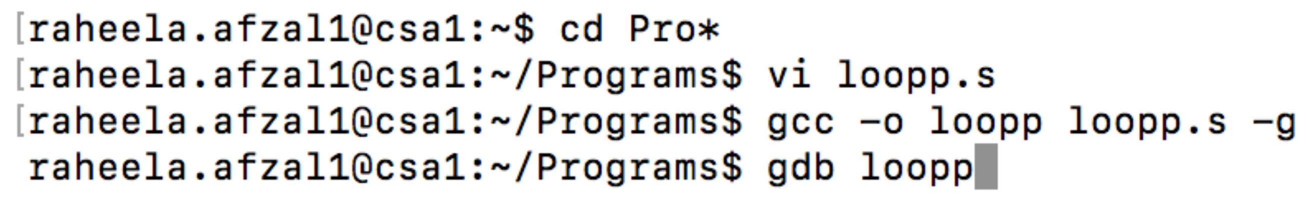
SOME COMMANDS

Command	Meaning	Example
r	Run the program	
q	Quit gdb	
c	Continue until next breakpoint or the end of program	
help	Print list of commands	
layout name	Change the layout of windows	`layout reg`

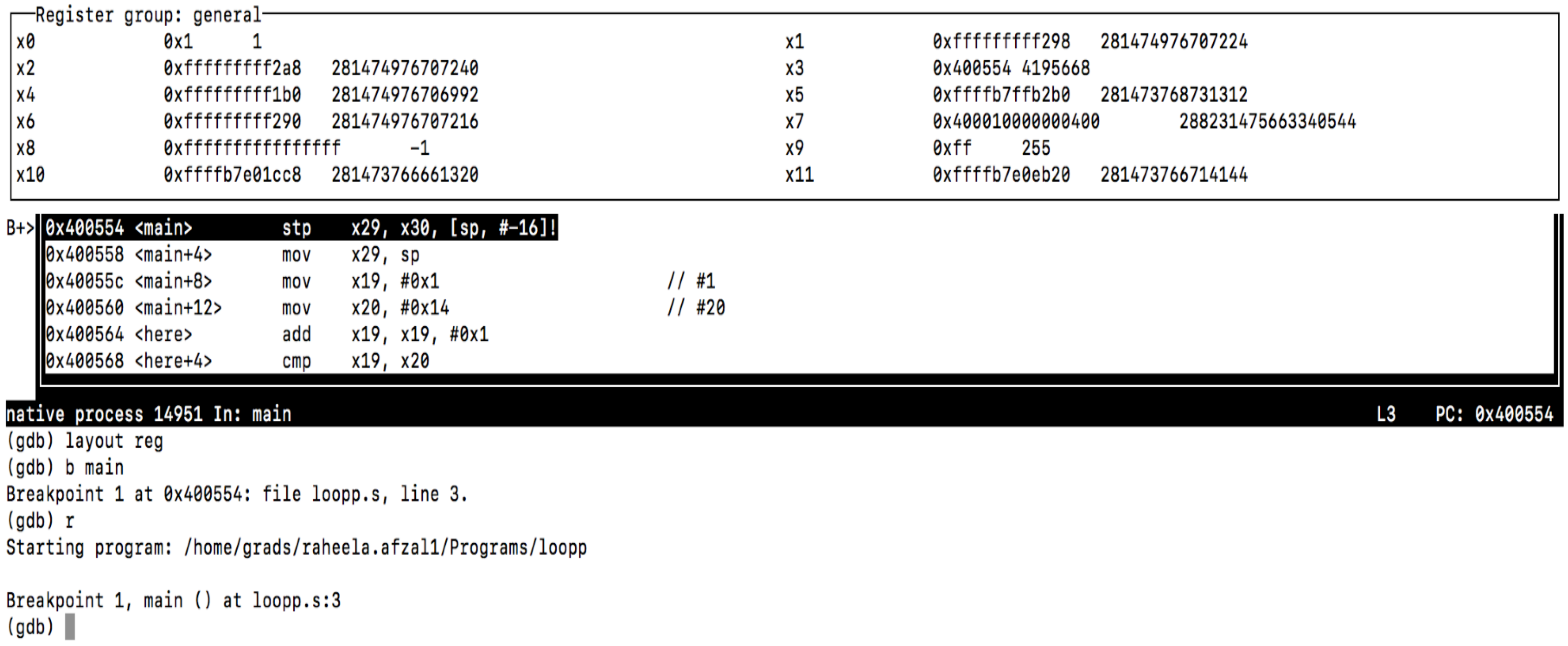
b label/line number	Set breakpoint	`b main`
		`b 9`

x/fmt addr	Examine memory at the address addr(in hex); fmt - i,x,d,u,s…	`x/i 0x440066`
p/fmt $xn	Print the contents of the register	`p/d $x20, p $x19`
display/fmt $xn	Auto display the contents of the register	`display/d $x23`
undisplay	Cancel all display requests	

GDB



```
[raheela.afzal1@csa1:~$ cd Pro*
[raheela.afzal1@csa1:~/Programs$ vi loopp.s
[raheela.afzal1@csa1:~/Programs$ gcc -o loopp loopp.s -g
 raheela.afzal1@csa1:~/Programs$ gdb loopp
```



```
┌─Register group: general──────────────────────────────────────────────────────
│x0             0x1      1                                      x1             0xfffffffff298   281474976707224
│x2             0xfffffffff2a8   281474976707240                x3             0x400554 4195668
│x4             0xfffffffff1b0   281474976706992                x5             0xffffb7ffb2b0   281473768731312
│x6             0xfffffffff290   281474976707216                x7             0x400010000000400        288231475663340544
│x8             0xffffffffffffffff       -1                     x9             0xff     255
│x10            0xffffb7e01cc8   281473766661320                x11            0xffffb7e0eb20   281473766714144

B+>│0x400554 <main>         stp    x29, x30, [sp, #-16]!
   │0x400558 <main+4>       mov    x29, sp
   │0x40055c <main+8>       mov    x19, #0x1                      // #1
   │0x400560 <main+12>      mov    x20, #0x14                     // #20
   │0x400564 <here>         add    x19, x19, #0x1
   │0x400568 <here+4>       cmp    x19, x20

native process 14951 In: main                                                                          L3    PC: 0x400554
(gdb) layout reg
(gdb) b main
Breakpoint 1 at 0x400554: file loopp.s, line 3.
(gdb) r
Starting program: /home/grads/raheela.afzal1/Programs/loopp

Breakpoint 1, main () at loopp.s:3
(gdb) 
```

MACRO PREPROCESSORS

Define a piece of text with a macro name: more readable code

```
define(coef, 23)
define(x_reg, x19)
..
add x20, x_reg, coef
```

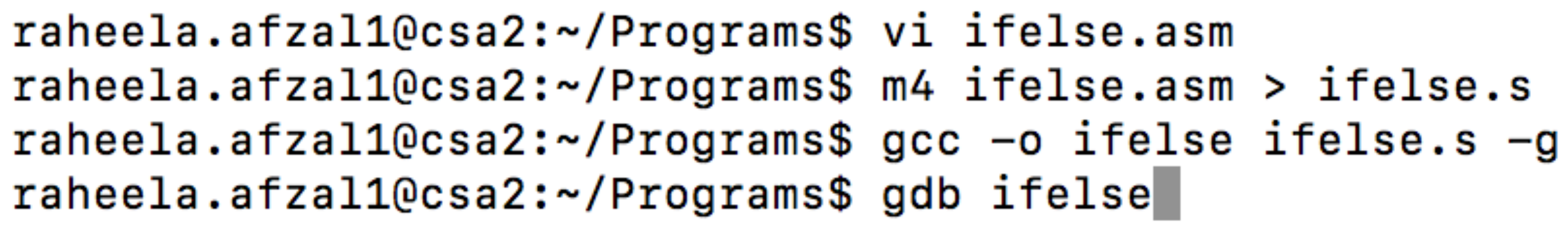
is expanded to

```
add x20, x19, 23
```

m4 is a macro processor, in the sense that it copies its input to the output, expanding macros as it goes.
Macros are either builtin or user-defined, and can take any number of arguments.

MACRO PREPROCESSORS

When using macros in your program, use the extension .asm followed by these instructions:



```
raheela.afzal1@csa2:~/Programs$ vi ifelse.asm
raheela.afzal1@csa2:~/Programs$ m4 ifelse.asm > ifelse.s
raheela.afzal1@csa2:~/Programs$ gcc -o ifelse ifelse.s -g
raheela.afzal1@csa2:~/Programs$ gdb ifelse
```

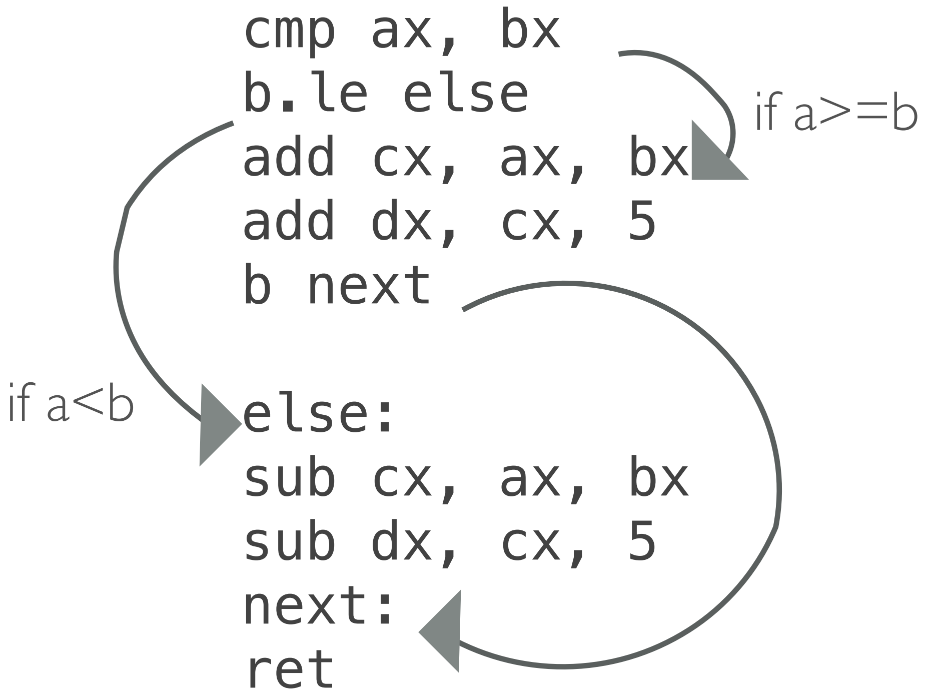
IF CONSTRUCTS

Assembly: ifelse.asm

```
define(ax, x19)
define(bx, x20)
define(cx, x21)
define(dx, x22) mov ax, 20
mov bx, 40

cmp ax, bx
b.le else
add cx, ax, bx
add dx, cx, 5
b next

else:
sub cx, ax, bx
sub dx, cx, 5
next:
ret
```



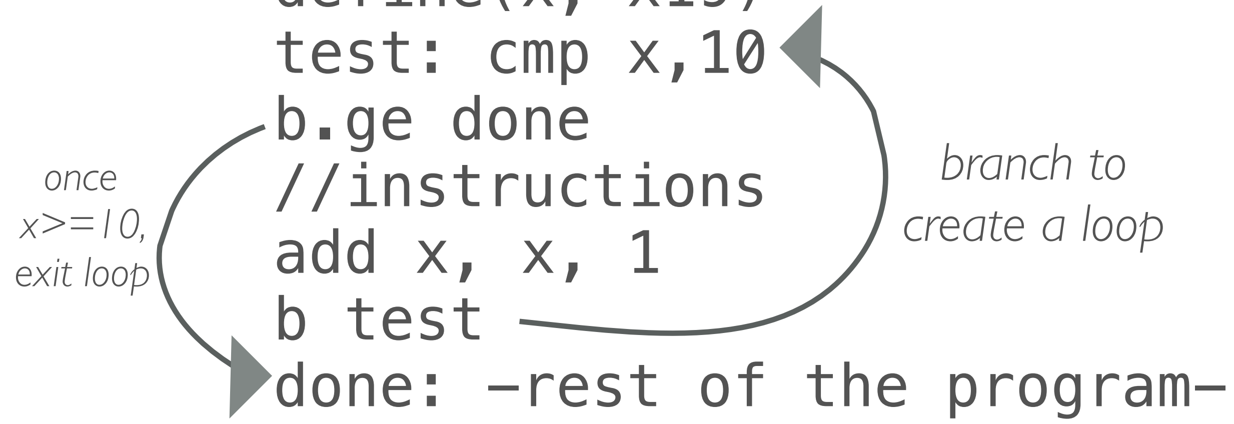
C code:

```
int a = 20, b=40;
    if(a>b){
   c = a + b;
   d = c + 5;
    } else {
   c = a - b;
   d = c - 5;
       }
```

Control Flow

LOOP

Assembly:



```
define(x, x19)
test: cmp x,10
b.ge done
//instructions
add x, x, 1
b test
done: -rest of the program-
```

C code:

```
int x;
x=0;
while(x<10){
//instructions

x++;
}
```

References

1. Computer Organization and Design by David A. Patterson & John L. Hennessy (ARM edition)
2. https://www.gnu.org/software/m4/manual/m4.html
3. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0473m/dom1359731152874.html