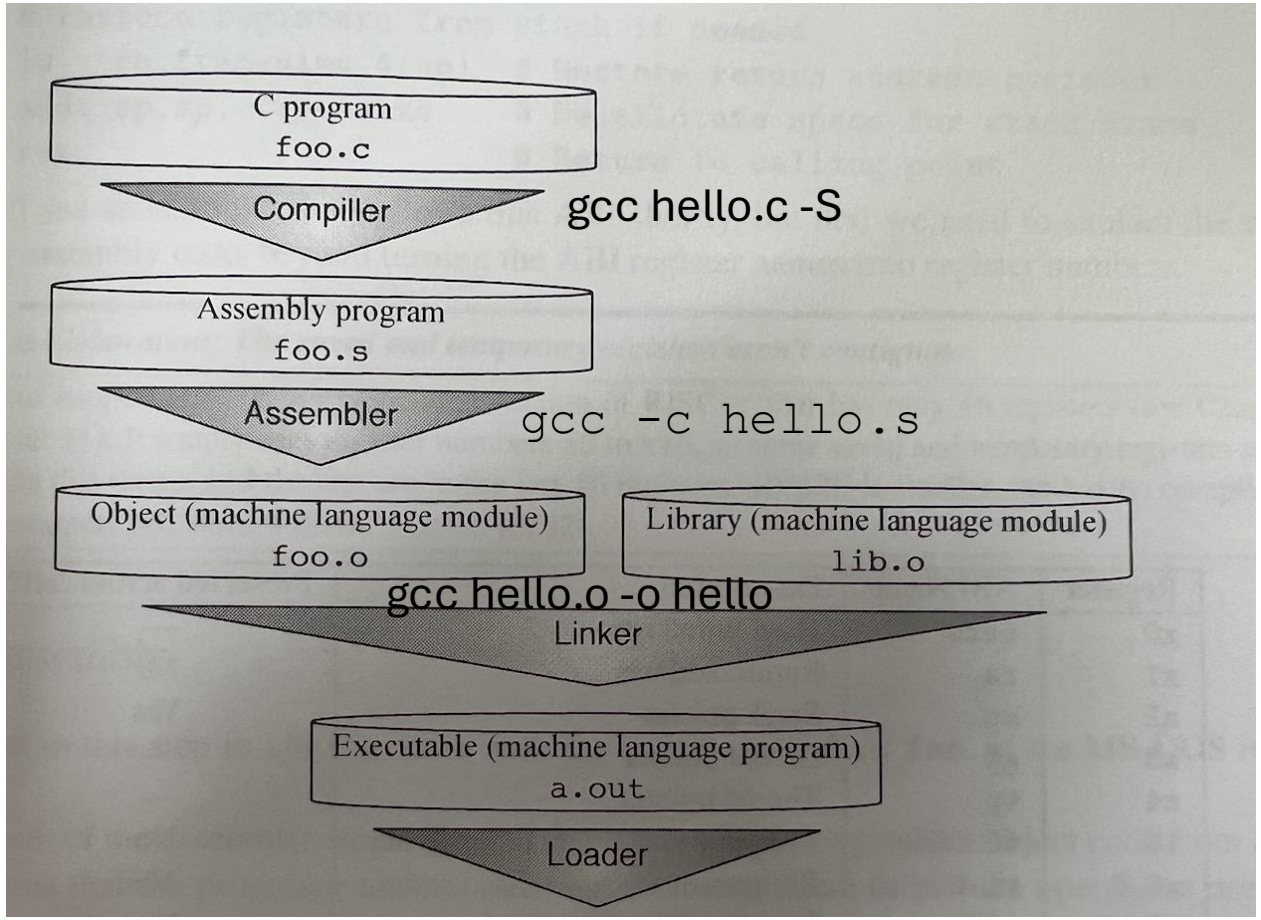
CSC 3210 Computer organization and programming

Chunlan Gao

Georgia State University

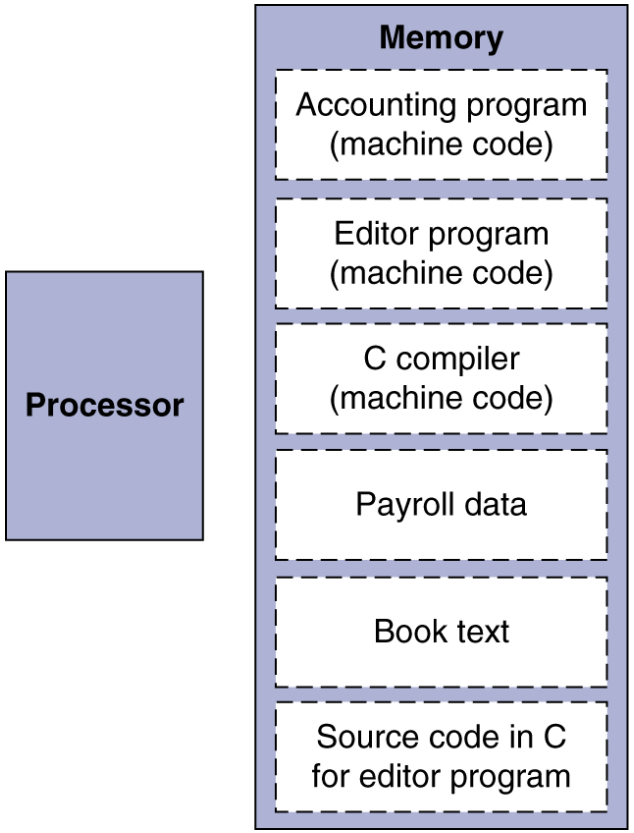
About the lab!



gcc -c program.c -o program.o

The -o option **allows custom naming** of the output file

Stored Program Computers



- Instructions represented in binary, just like data

Instructions and data stored in memory

- Programs can operate on programs
  e.g., compilers, linkers, …
- Binary compatibility allows compiled programs to work on different computers
  - ❖Standardized ISAs

Logical Operations

- Instructions for bitwise manipulation

Logical operations	C operators	Java operators	RISC-V instructions
Shift left	<<	<<	sll, slli
Shift right	>>	>>>	srl, srli
Shift right arithmetic	>>	>>	sra, srai
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR	|	|	or, ori
Bit-by-bit XOR	^	^	xor, xori
Bit-by-bit NOT	~	~	xori

FIGURE 2.8 C and Java logical operators and their corresponding RISC-V instructions. One way to implement NOT is to use XOR with one operand being all ones (FFFF FFFF FFFF $FFFF_{hex}$).

Shift Operations(I type)

funct7	immediate	rs1	funct3	rd	opcode
0	4	19	1	11	19

immed: how many positions to shift

Shift left logical:

Shift left and fill with 0 bits
`slli` by *i* bits multiplies by 2^i

Shift right logical:

Shift right and fill with 0 bits
`srli` by *i* bits divides by 2^i
(unsigned only)

Example

slli x11, x19, 4

if register x19 contained
$00000000\ 00000000\ 00000000\ 00001111_{two}$
$= 15_{ten}$

and the instruction to shift left by 4 was executed, the new value would be:

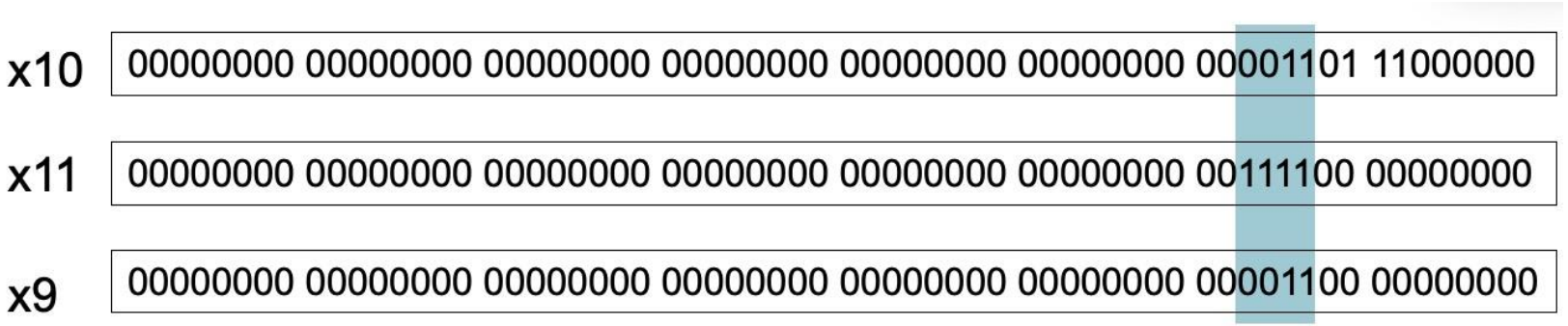
00000000 00000000 00000000 11110000two

$= 240_{ten}$

AND Operations

	1	0	0	1
AND	0	1	0	1
	0	0	0	1

- Useful to mask bits in a word
  - Select some bits, clear others to 0b
  - `and x9,x10,x11`

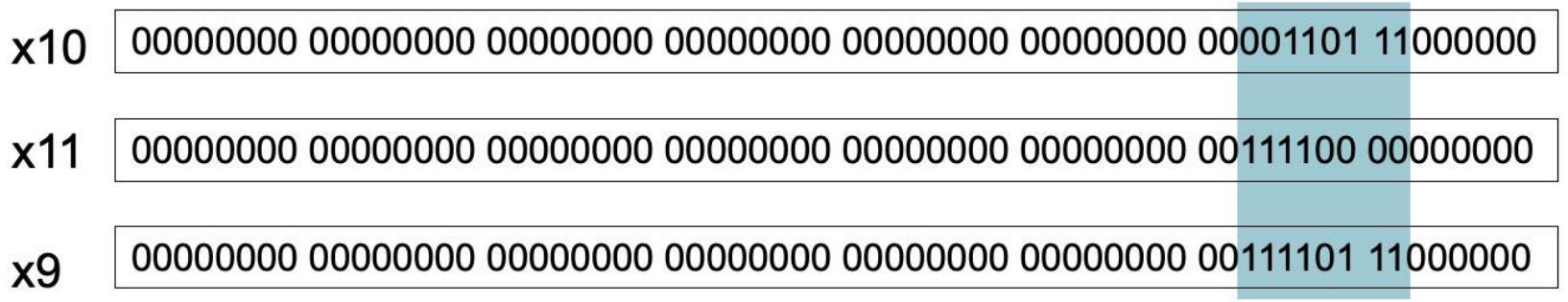


OR Operations

	1	0	0	1
OR	0	1	0	1
	1	1	0	1

- Useful to include bits in a word
  - Set some bits to 1, leave others unchanged

```
or x9,x10,x11
```

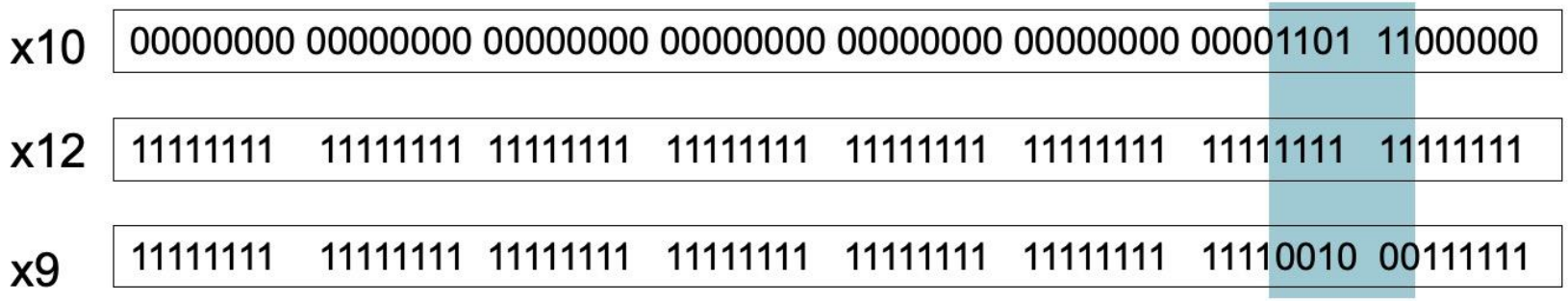


XOR Operations

	1	0	0	1
XOR	0	1	0	1
	1	1	0	0

- Differencing operation
  - Set some bits to 1, leave others unchanged

```
xor x9,x10,x12  // NOT operation
```



Instructions for Making Decisions

- What distinguishes a computer from a simple calculator is its ability to make decisions. Based on the input data and the values created during computation, different instructions execute.

Conditional Operations

- Branch to a labeled instruction if a condition is true
  - Otherwise, continue sequentially

- `beq rs1, rs2, L1`
  - if (rs1 == rs2) branch to instruction labeled L1

- `bne rs1, rs2, L2`
  - if (rs1 != rs2) branch to instruction labeled L2

```
.L4:
        movl    -4(%rbp), %eax
        andl    $1, %eax
        testl   %eax, %eax
        jne     .L3
        movl    -4(%rbp), %eax
        movl    %eax, %esi
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
.L3:
        addl    $1, -4(%rbp)
.L2:
        cmpl    $20, -4(%rbp)
        jle     .L4
        movl    $10, %edi
        call    putchar
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

Compiling if-then-else into Conditional Branches

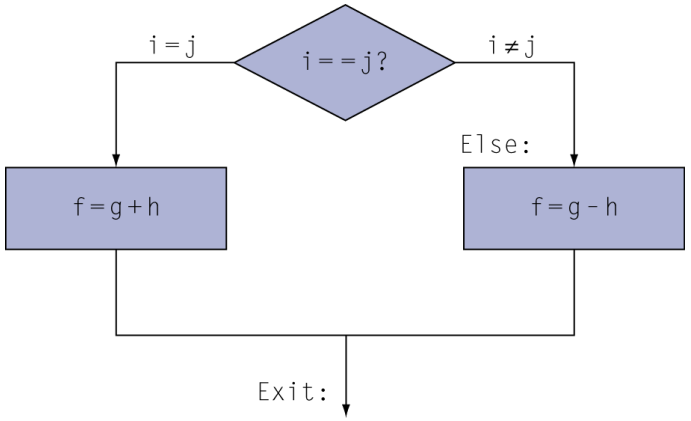
- C code:

```
if (i==j)  f = g+h;
else f = g-h;
```

f	g	h	i	j
X19	X20	21	X22	X23

- Compiled RISC-V code:

```
      bne x22, x23, Else
      add x19, x20, x21
      beq x0,x0,Exit // unconditional
Else: sub x19, x20, x21
Exit:
```



Compiling Loop Statements

- C code:

```
while (save[i] == k)
        i += 1;
```

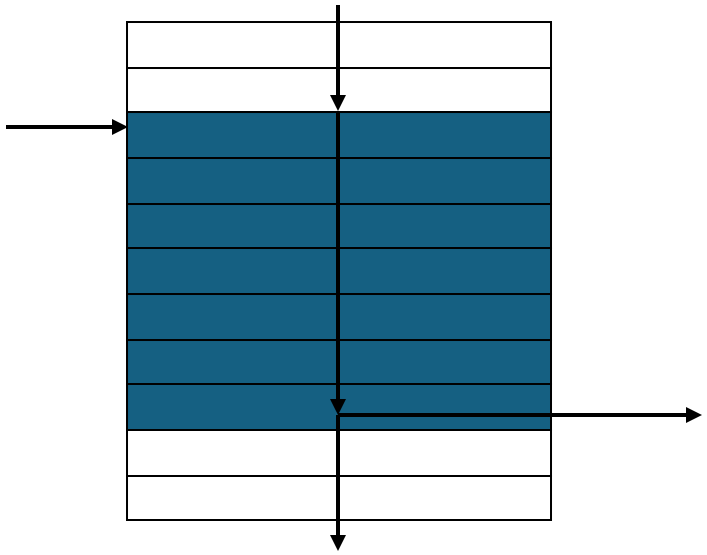
  - i in x22, k in x24, address of save in x25

- Compiled RISC-V code:

```
Loop: slli x10, x22, 2      // regX10 = i*4
      add  x10, x10, x25    // X10= address of save[i]
      lw   x9, 0(x10)       // temp reg X9 =save[i]
      bne  x9, x24, Exit //  go to Exit if save[i] ≠ k
      addi x22, x22, 1   //   i = i+1
      beq  x0, x0, Loop  //   go to the loop table
Exit: …
```

Basic Blocks

- A basic block is a sequence of instructions with
  - No embedded branches (except at end)
  - No branch targets (except at beginning)



A compiler identifies basic blocks for optimization.
An advanced processor can accelerate execution of basic blocks.