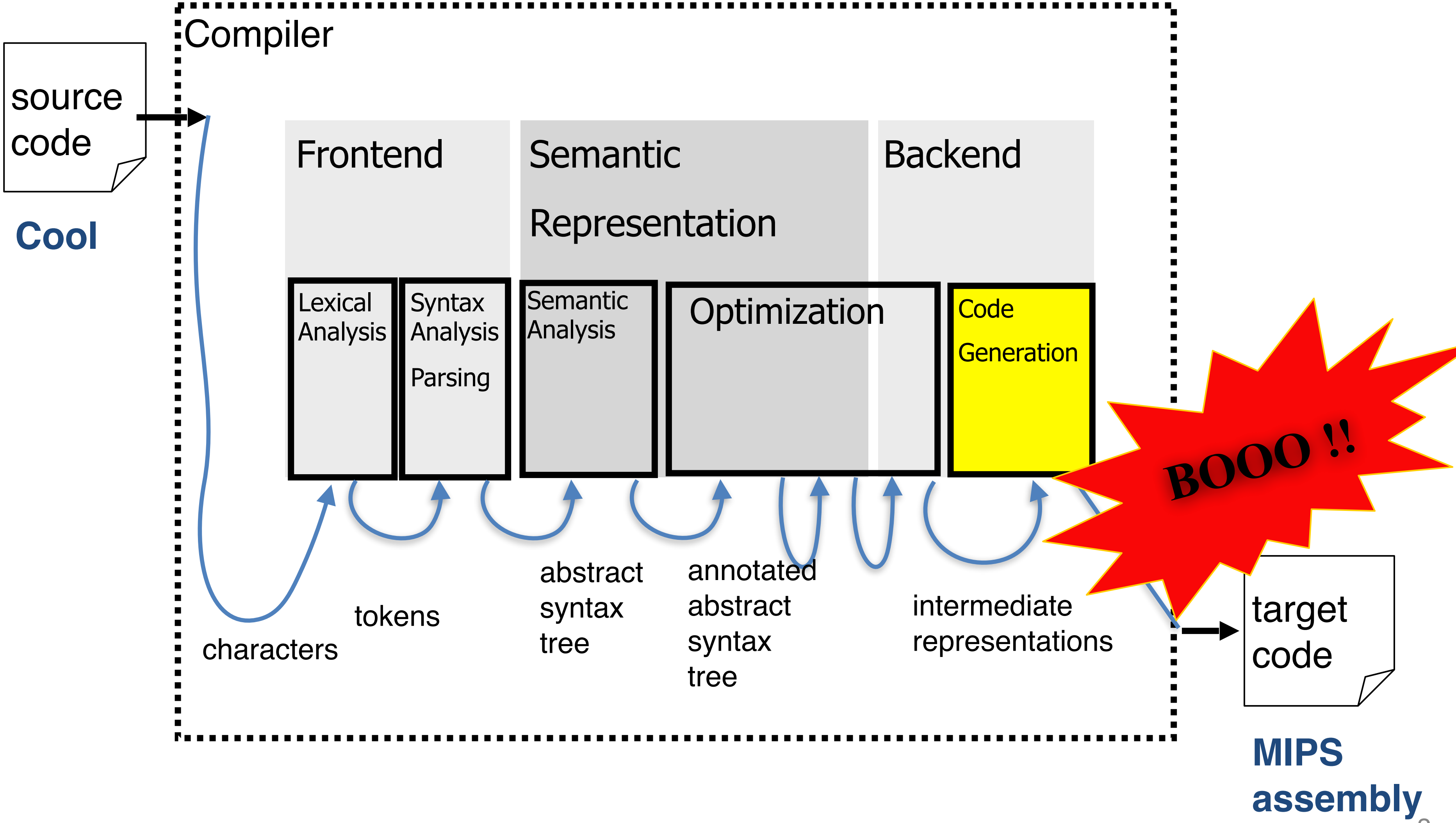
ECS652: Compilers (Spring 2018)

Tutorial 08

MIPS Architecture and Assembly

Greta Yorsh

PA3: Backend


source code
Cool
Compiler
Frontend
Semantic Representation
Backend
Lexical Analysis
Syntax Analysis
Parsing
Semantic Analysis
Optimization
Code Generation
BOOO !!
characters
tokens
abstract syntax tree
annotated abstract syntax tree
intermediate representations
target code
MIPS assembly

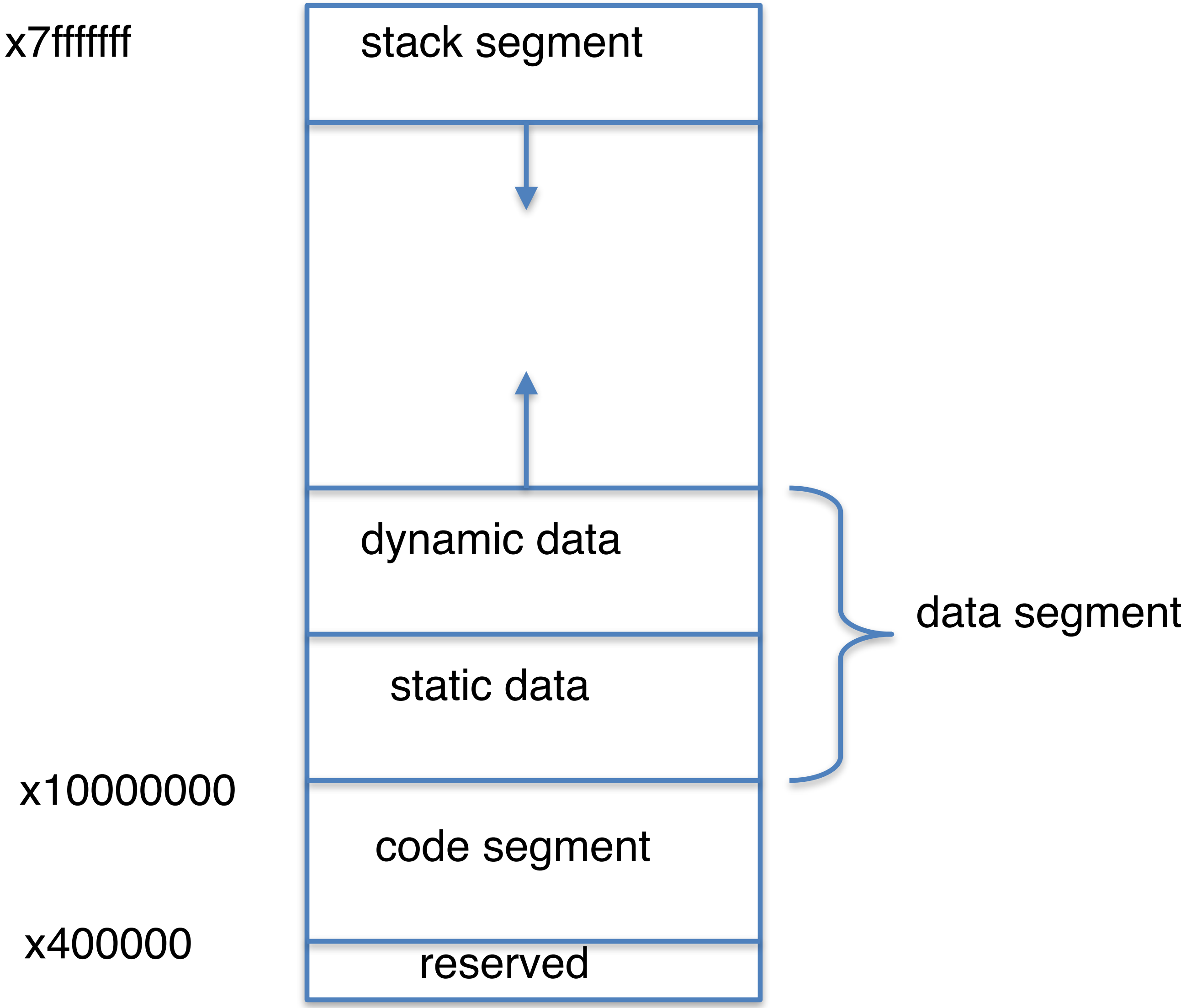
Overview

- Memory
- Registers
- Instructions
- Examples

Memory

- Store data and instructions
- Partitioned into bytes, words
- Virtual memory expands physical memory size
- Bounded by machine address size (32 bit)

Memory


x7fffffff
stack segment
dynamic data
data segment
static data
x10000000
code segment
x400000
reserved

Gulliver's Travels by Jonathan Swift

- **Little Endian**: store the little end of the number (least significant bits) first
- **Big Endian**: store the big end of the number first

- Cool MIPS is little-endian

Example: little endian

0x1000000007	0x00
0x1000000006	0x00
0x1000000005	0x7f
0x1000000004	0xfe
0x1000000003	0x70(‘p’)
0x1000000002	0x65(‘l’)
0x1000000001	0x65(‘e’)
0x1000000000	0x48(‘H’)

3277 = 0x00007ffe

Help

Registers

- General-purpose: 32 registers
- Special-purpose: a few more

General-purpose registers

Name	Software Name	Usage
$0		Always 0
$1	at	Reserved for the assembler
$2…$3	v0-v1	Return values
$4…$7	a0-a3	First arguments
$16…$23	s0-s7	Callee-saved registers
$24…$25	t8-t9	Caller-saved registers
$29	sp	Stack pointer
$30	fp	Frame pointer
$31	ra	The return address

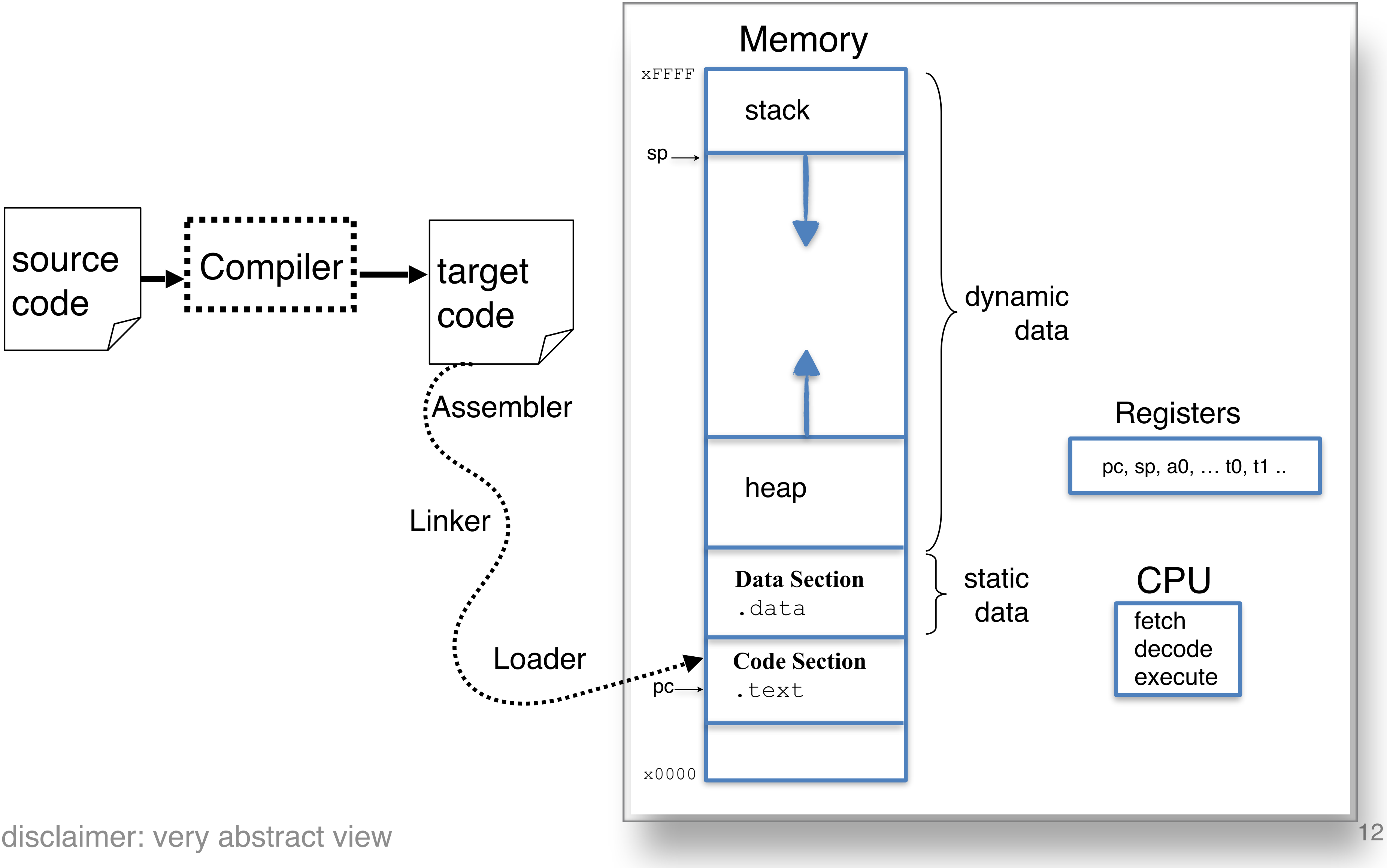
Special Purpose Registers

Name	**Usage**
PC	Program Counter
HI	The most significant 32 bits after multiply and divide
LO	The least significant 32 bits after multiply and divide

What is a CPU?

- A fast interpreter for machine code (sort of)

Hardware



*disclaimer: very abstract view

RISC vs. CISC machines

Feature	**RISC**	**CISC**
Registers	32	6, 8, 16
Register classes	One	Some
Arithmetic operands	Registers	Memory+Registers
Instructions	3-addr	2-addr
Addressing modes	r M[r+c] (l,s)	several
Instruction length	32 bits	Variable
Side-effects	None	Some
Instruction-cost	“Uniform”	Varied

Assembly code

- Allows symbolic names of registers and machine instructions
- Simplifies the task of the compiler writer
- Easier to debug the compiler
- One to one translation into machine code
- But may require two passes on the assembly

Instructions: load and store

- load from memory to register
- store from register to memory
- load immediate

Instruction	**Meaning**
li $v0 4	$v0 := 4
la $a0 msg	$a0 := address of msg
lw $t0 x	$t0 := x
sw $t0 y	y := $t0

Instructions: load and store

- la vs. li
    - label represents a fixed memory address after assembly
    - la is a special case of load immediate
- lw vs. la
    - x is at address 10 and contains 2
    - la $a0 x   **$a0 := 10**
    - lw $a0 x   **$a0 := 2**
- lw $t0 8($sp)

Instructions: arithmetic and logic operations

- perform the operation on data in 2 registers
- store the result in a 3rd register

Instruction	Meaning
add $t0 $t3 $t4	$t0 := $t3 + $t4
sub $t0 $t3 $t1	$t0 := $t3 - $t4
mul $t0 $t3 35	$t0 := $t3 * 35

- variants for unsigned: addu, subu, mulu

Instructions: jump and branch

Instruction	Meaning
j label	jump to instruction at label
jal my_proc	jump and link start procedure my_proc $ra holds address of instruction following the jal
jr $ra	jump register return from procedure call puts $ra value back into the PC
beq $t0 $t1 label	if ($t0 = $t1) goto label
bneq $t0 $0 label	if ($t0 != $0) goto label
bltz $t0 label	if ($t0 < 0) goto label

bltu, bltuz, bgtz, …

Example

```
while (x < 0) {
     x=x-1;
}
```

```
         lw $t0, x
loop_1:  bgez $t0 loop_1_out
         subi $t0 $t0 1
         b loop_1
loop_1_out:
```

(warning: code shown is a naïve code :-)

Instructions

- Operations
  - load and store
  - arithmetic and logical operations
  - jump and branch
  - specialized instructions
  - floating-point instructions and registers
- Instruction formats
  - register
  - immediate
  - jump

Addressing modes

- Immediate: value built in to the instruction
- Register: register used for data
- Memory referencing
    - used with load and store instructions
    - label: fixed address built in to the instruction
    - indirect: register contains the address
        - base addressing - field of a record
        - indexed addressing - element of an array

System calls

Service	Code	Arguments	Result
print integer	1	\$a0=integer	console print
print string	4	\$a0=string address	console print
read integer	5		\$a0=result
read string	8	\$a0=string address \$a1=length limit	console read
exit	10		end of program

Hello World

```
       .text                   # data segment
       .globl __start
__start:                       # execution starts here
       la $a0 str              # put string address into a0
       li $v0 4                #
       syscall                 # print
       li v0  10               #
       syscall                 # au revoir...

       .data                   # data segment
str:   .asciiz  "hello world\n"
```

Riddle

```
     .data
endl:  .asciiz  “\n”

     .text
print_int:
     li  $v0 1
     syscall
     jr  $ra
endline:
     la  $a0 endl
     li  $v0 4
     syscall
     jr  $ra
__start:
     li  $a0 42
     jal print_int
     jal endline
     li  $a0 2027
     jal print_int
     jal endline
```

Summary

- Understand basic ideas in MIPS architecture
- Can you manually convert a simple code into a naïve MIPS assembly ?

BOOO ??