COMP 273: MIPS & MARS

Gordon Krieger

February 5, 2020

Assembly language: old killer technology

The Terminator 6502 Assembler

register conventions

0	$zero	constant zero
1	$at	reserved for assembler
2-3	$v0 - $v1	return values, syscalls
4-7	$a0 - $a3	arguments
8-15	$t0 - $t7	temporary values
16-23	$s0 - $s7	“saved” values
24-25	$t7- $t8	more temporaries
26-27	$k0 - $k1	reserved for kernel
28	$gp	global pointer
29	$sp	stack pointer
30	$fp	frame pointer
31	$ra	return address

register conventions: impossible to change

0	$zero	constant zero
1	$at	reserved for assembler
2-3	$v0 - $v1	return values, syscalls
4-7	$a0 - $a3	arguments
8-15	$t0 - $t7	temporary values
16-23	$s0 - $s7	“saved” values
24-25	$t7- $t8	more temporaries
26-27	$k0 - $k1	reserved for kernel
28	$gp	global pointer
29	$sp	stack pointer
30	$fp	frame pointer
31	$ra	return address

register conventions: preserve across calls

0	$zero	constant zero
1	$at	reserved for assembler
2-3	$v0 - $v1	return values, syscalls
4-7	$a0 - $a3	arguments
8-15	$t0 - $t7	temporary values
16-23	$s0 - $s7	"saved" values
24-25	$t7- $t8	more temporaries
26-27	$k0 - $k1	reserved for kernel
28	$gp	global pointer
29	$sp	stack pointer
30	$fp	frame pointer
31	$ra	return address

register conventions: commonly used

0	$zero	constant zero
1	$at	reserved for assembler
2-3	$v0 - $v1	return values, syscalls
4-7	$a0 - $a3	arguments
8-15	$t0 - $t7	temporary values
16-23	$s0 - $s7	“saved” values
24-25	$t7- $t8	more temporaries
26-27	$k0 - $k1	reserved for kernel
28	$gp	global pointer
29	$sp	stack pointer
30	$fp	frame pointer
31	$ra	return address

assembler directives

- These are instructions for the assembler that are not translated into machine code
- used in the data portion of your .asm file
- anything beginning with a period is an assembler directive: `.word` `.byte` `.double` `.float` `.asciiz` `.data` *etc*
- type a period in MARS for a full menu of assembler directives

typical assembler directive

```
hello: .asciiz "Hello World!"
```

- Store a null-terminated string into memory
- the label "hello" points to the starting address
- you'll need the label later to refer to the string in your code (recall that assembly language has no variables)

assembler directives: storing numbers

```
answer: .word 42
```

store a value using 32 bits in two's complement format

```
shorter_answer: .byte 42
```

store a value using 8 bits in two's complement format

```
almost_A: .float 84.999
```

store a value in IEEE single precision format

▸ full list of MIPS assembler directives

assembler directives: dire warning

Warning: MIPS is untyped!

- it's up to you to remember whether a particular value is meant to represent an integer, a floating point number, part of a string...
- most high-level languages do this work for you, and complain if you make a mistake
- MIPS will just play along and give you garbage

syscalls

- Syscalls request services from the OS, typically handling input or output
- Each service has its own identifying number (load into register $v0)
- Some services require additional arguments (often loaded into $a registers, varies by syscall service)

Typical syscall:

```
li $v0, 4          # syscall code for print string
la $a0, hello      # load address of string into $a0
syscall
```

▸ full list of MIPS syscalls

MIPS instructions

... *we will cover these as we go*

arithmetic / logical

`add, sub, mult, div, and, or, sll, slt ...`

data handling

`lw, sw ...`

conditional branching

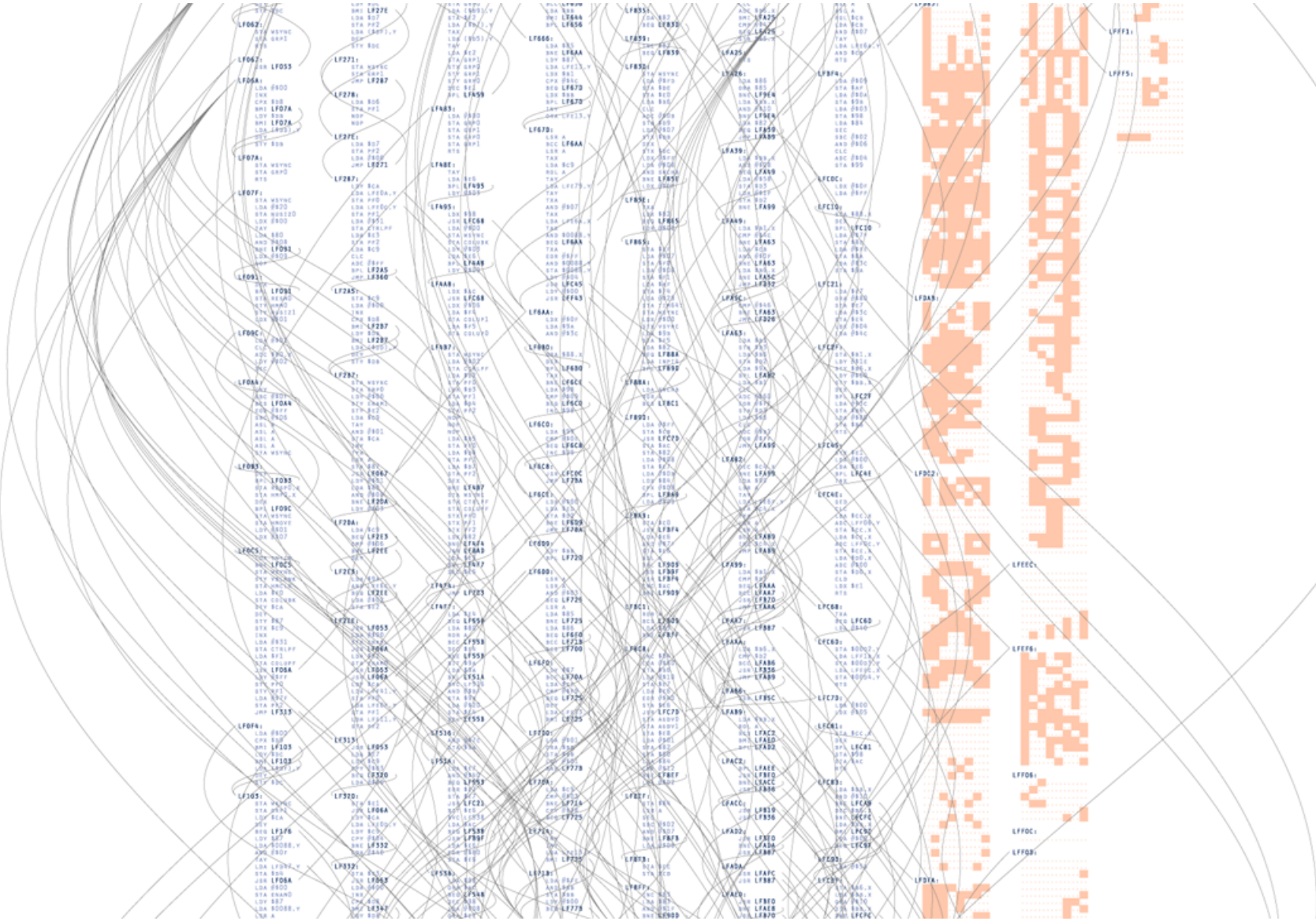
`bne, beq, ...`

unconditional jumps

`j, jal, jr`

▸ MIPS instruction reference ▸ MIPS cribsheet

Pac-Man Assembler code (Atari 2600 version)



https://benfry.com/distellamap