cpu name:

yo

language name:

gurt

created by:
brandon gaona & zidan kazi
i pledge my honor that i have abided by the stevens honor system
===

syntax:

all gurt instructions are:

- completely lowercase
- separated by spaces
- intentionally minimal
- designed for the *yo* 8-bit instruction format

add

performs register addition

- computes rs + rt and stores the result in rd
- format: add rd rs rt
- opcode: 00

sub

performs register subtraction

- computes rs - rt and stores the result in rd
- format: sub rd rs rt
- opcode: 01

load

loads a value from memory into a register

- computes address = rb + offset imm and loads mem[address] into rd
- format: load rd rb imm
- opcode: 10

store

stores a register value into memory

- computes address = rb + imm and writes rs into mem[address]
- format: store rs rb imm
- opcode: 11

==

registers

the yo cpu has four general-purpose registers
r0 r1 r2 r3
each register is encoded using 2 bits:

r0 = 00
r1 = 01
r2 = 10
r3 = 11

==

running the assembler

to assemble a gurt program:

1. create a .gurt file that contains a .text section for instructions and a .data section for initial memory values.
   - all instructions must be lowercase and space-separated.
2. make sure all gurt instructions follow the formats listed in this manual
   - (add, sub, load, store) and that all registers are written as r0–r3
3. place your .gurt file in the same directory as assembler.py
4. open a terminal in that directory and run:
   - python assembler.py filename.gurt
5. the assembler will automatically do the following:

  - clean any old memory image files
  - read your .gurt file
  - separate the .text and .data sections
  - convert each gurt instruction into its 8-bit machine code
  - convert each data value into a memory word
  - write two new files:
    - instruction_mem.hex: contains your compiled machine instructions
    - data_mem.hex: contains the initial memory contents from the .data section
6. right-click the instruction memory component -> “load image” -> select instruction_mem.hex
7. right-click the data memory component -> “load image” -> select data_mem.hex
8. reset the cpu and begin execution

==

architecture

the yo cpu is an 8-bit single-cycle processor

it contains:

- a program counter
- instruction memory
- an instruction splitter
- a control unit
- a register file with four general-purpose registers
- an alu that performs add and sub
- a data memory

all instructions are 8 bits wide

the cpu executes one instruction per clock cycle
the program counter increments by 1 after each instruction

the cpu supports:
add sub load store

==

binary encoding

each yo instruction is 8 bits wide. the instruction fields are:

opcode: 2 bits

- allows up to 4 instructions

<u>register fields (rd rs rt rb)</u>: 2 bits each

- because the cpu has 4 registers

<u>immediate field (for load and store):</u> 2 bits

- allows offsets from 0–3

add, sub

- format: add rd rs rt
- format: sub rd rs rt

encoding structure:

opcode (2 bits) + rd (2 bits) + rs (2 bits) + rt (2 bits)

example:
add r1 r2 r3
opcode = 00
rd = 01
rs = 10
rt = 11
machine code:
00 01 10 11

load

- format: load rd rb imm

encoding structure:
opcode (2 bits) + rd (2 bits) + rb (2 bits) + imm (2 bits)

example:
load r1 r0 2
opcode = 10
rd = 01
rb = 00
imm = 10
machine code:
10 01 00 10

store

- format: store rs rb imm

encoding structure:
opcode (2 bits) + rs (2 bits) + rb (2 bits) + imm (2 bits)

example:
store r3 r0 2
opcode = 11
rs = 11
rb = 00
imm = 10
machine code:
11 11 00 10

==

jobs

brandon:
built the cpu and added the fun components, worked on the demo

zidan:
designed the gurt language, worked on the demo program, and wrote the user manual