NAME

asm - simple assembler for the CSE471 subset of the MIPS instruction set.

SYNOPSIS

asm [file]

DESCRIPTION

asm translates assembly code (either in the file specified by the single argument, or from standard input) into machine code. The program produces four output files named: prog.bin, prog.hex, data.bin, and data.hex.

These files can be used to initialized memory arrays using the Verilog-XL commands $readmemb and $readmemh.

ASSEMBLY CODE SYNTAX

The MIPS architecture specifies 32 registers. These are named $0, $1, $2, ..., $31. There are five major forms for assembly instructions:

Memory instructions (includes: lw, sw)

```
lw $3,2000($4)
sw $3,-0xeff($4)
sw $3,-0b1011($4)
lw $3,d_one
```

Register instructions (includes: and, sub, or, and, xor, slt)

```
add $3,$4,$5
```

Immediate instructions (includes: addi, ori, andi, xori, slti)

```
addi $3,$4,-0x73
```

Branch instructions (includes: beq, bne)

```
beq $3,$4,loop
```

Jump instructions (includes: j, jal)

```
j loop
```

There are two that defy pigeon-holing:

lui, which takes one register and an immediate, and jr, which takes one register.

Instructions can be labeled; so can storage locations in the data segment. Labels are specified by a alphanumeric string followed by a colon written before the instruction. The assembler converts referenced labels into the appropriate machine code offset.

There are three other directives for the assembler. The .text string states that the lines that follow represent instructions that belong in the instruction memory; the .data string states that the following .word strings (used to reserve 4 bytes of memory) are to be allocated in the data memory.

EXAMPLE

```
     .text
     lw      $2,four
     lw      $3,ubound
     lw      $6,one
     add     $4,$0,$0
     add     $5,$0,$0
top: slt     $8,$4,$3
     beq     $8,$0,loop
     add     $7,$5,$6
     sw      $7, 0($4)
     add     $4,$4,$2
     add     $5,$6,$7
     sw      $5, 0($4)
     add     $4,$4,$2
     add     $6,$7,$5
     sw      $6, 0($4)
     add     $4,$4,$2
     beq     $0, $0, top
loop: beq     $0, $0, loop

     .data
four:   .word 0x00000004
ubound: .word 0x00000024
one:    .word 0x00000001
```

"SEE ALSO"

verilog(1)

AUTHOR

Steve Burns (burns@cs.washington.edu)