GREEN CARD ISA E INFORMAÇÕES SOBRE O USO

Davi Manzini, GRR20245482

Desenvolvi minha ISA com 3 tipos de instrução possíveis: R, I e J. Devido às limitações do trabalho, cada instrução possui apenas 12 bits de informação. A instrução R possui os bits de 0 a 3 como opcode, 4 a 7 como RB (registrador) e 8 a 11 como RD/RA (registrador) - nesse caso o registrador destino é também um dos registradores usados na operação da instrução. A instrução do tipo I também possui os bits de 0 a 3 como opcode e os bits de 8 a 11 como RD, mas dessa vez os bits de 4 a 7 representam um imediato. Por fim, a instrução do tipo J possui bits de 0 a 3 como opcode e bits de 4 a 11 como imediato - essa instrução é apenas usada para o JAL (para ser possível serem realizados saltos maiores).

No meu banco de registradores, há 16 registradores, uma vez que tenho 4 bits para eles. o R0 e R1 são registradores fixos e não devem ser usados livremente. O R0 é fixo com valor 0 e é usado na instrução BEQ e o R1 recebe sempre PC + 1 e é usado para salvar a próxima instrução quando é usada a instrução JAL. Os outros registradores, de R2 a R15, são de livre uso.

Implementei o PC usando um registrador que, dependendo da instrução, pode receber PC + 1, PC + IMM ou diretamente o valor de um registrador (no caso do JR).

Importante ressaltar que a instrução BPOS é como se fosse uma pseudo instrução BGE, mas devido à limitação de bits ela é um pouco diferente. Como na BGE um valor x1 tem que ser maior ou igual a outro valor x2, pode-se dizer que x1 - x2 tem que ser maior que 0. Por conta disso, a instrução BGE clássica é dividida em uma subtração usando SUB e depois na instrução BPOS (do tipo I) que analisa se o registrador é maior ou igual a 0 (o que aciona o túnel POSITIVO no circuito) - caso seja, o desvio é tomado.

Além disso, listo algumas outras informações importantes a serem relatadas: novamente devido à limitação de bits da instrução, para usar as instruções LW e SW, é necessário na instrução anterior calcular de onde será feito o *load* ou onde será feito o *store*.

Em relação ao programa de testes: o circuito entregue possui uma memória ROM como unidade de controle. Seu conteúdo foi preenchido através da tabela de sinais de controle de todas as instruções (montada por mim e anexada também nesse repositório do GitHub. Eu peguei a linha da tabela referente a cada instrução e montei um número em binário de 13 bits (referente a todos os bits dos sinais de controle). Depois disso, fiz a conversão de cada um desses números para hexadecimal e coloquei na posição relativa ao OPCODE da operação dentro da ROM (uma vez que o OPCODE entra como endereço na ROM). Além disso, na memória de instruções do circuito entregue, há as instruções para executar o programa de teste (soma de vetores) indicado nas instruções do trabalho, para que o professor consiga

fazer o teste mais facilmente. De qualquer modo, também anexado nesse repositório, está todo o código em Assembly que traduzi do código em C, a tradução para binário e uma ainda posterior tradução para hexadecimal também. Noto que a tradução de Assembly para binário foi feita por mim através da minha tabela de sinais de controle/OPCODE (a tradução para hexadecimal foi feita usando conversor online instrução por instrução. OBS: tive que fazer o código da soma de vetores em Assembly um pouco diferente da proposta em C: minhas instruções que possuem salto (com exceção do JAL) possuem apenas 4 bits de imediato, o que não seria suficiente para o salto do primeiro for. Por isso, dividi o primeiro for em 2, para conseguir usar minhas instruções de forma correta.

Por fim, o professor verá que há algumas saídas conectadas a diversos fios diferentes ao longo do circuito entregue. O intuito delas é facilitar os testes e também foram usadas por mim na hora de debugar, para poder ser checado quais valores estavam realmente passando pelos fios.

INSTRUÇÕES E OPERAÇÕES:

TIPO R: [11:8] RD, [7:4] R2, [3:0] OPCODE

TIPO I: [11:8] RD, [7:4]IMM, [3:0] OPCODE

TIPO J: [11:4] IMM, [3:0] OPCODE

OBS: Nas instruções do tipo R e I, o registrador destino (RD) também é o primeiro registrador da operação (Ex: add r2, r3 = r2 recebe r2 + r3)

OBS: A instrução J é apenas usada pela operação JAL, para ser possível fazer um maior salto.

OPERAÇÕES:

ADD - TIPO R
SUB - TIPO R
MUL - TIPO R
AND - TIPO R
OR - TIPO R
BEQ - TIPO I
JAL - TIPO J
BPOS - TIPO I
SW - TIPO R

LW - TIPO R
ADDI - TIPO I
JR - TIPO J
HALT