

Primeiro Trabalho Prático

IDENTIFICAÇÃO:

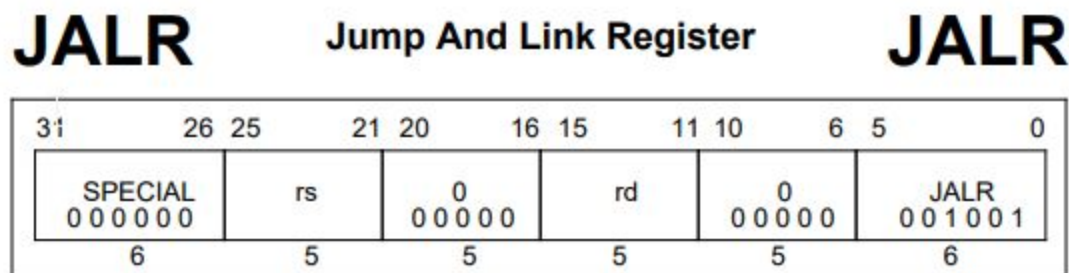
Nomes:

- Leonardo Vianna - 00274721
- Gabriel Stepien - 00265035
- Andy Ruiz - 00274705
- Mauricio Holler - 00273108

Professor: Luigi Carro

Modificações

JALR rd rs:



JALR é uma *special instruction*, para a identificar foi criado dois AND's, um NOR e um MUX para enviar os seis primeiros bits para o controle ao invés dos seis últimos e a ROM foi modificada para suportar a nova função.

Monociclo:

O monociclo recebeu um bit de controle a mais chamado "**JALR**" e dois MUX's foram criados, um para enviar para o PC o valor colocado no registrador "**RS**" e outro para salvar o valor do PC+4 no registrador "**RD**" que são ligados com o bit "**JALR**".

Multiciclo:

O multiciclo recebeu os mesmos MUX's que o monociclo, mas o PC não é incrementado porque o PC já estará na próxima posição e também recebeu um estado novo "**B**" que contém o ligamento do novo bit de controle e dos respectivos bits para haver uma escrita no PC e outra escrita nos registradores. A ordem dos estados fica: $0 \rightarrow 1 \rightarrow B \rightarrow 0$.

Pipeline:

O pipeline recebeu os mesmos MUX's, entretanto recebeu seis buffers e um sinal de controle a mais. Dois dos cinco registradores são para repassar o sinal de controle "**JALR_PC**" do estado EX para o sinal "**JALR_WB**" que está no estado WB que são dois estados a frente. Os outros três registradores são para manter o PC desejado até o estado WB para que seja escrito no registrador desejável, aqui o PC não é incrementado porque o PC já estará na próxima posição, houve uma mudança no controle para ligar os bits para escrever no registrador, escrever no PC e ligar o "**JALR_PC**" que fará a escrita no PC corretamente e repassará o bit 1 para o "**JALR_WB**" que fará a escrita no registrador desejado do valor do PC antes da escrita do JALR_PC.

Todas as três formas de MIPS foram testadas com o mesmo programa fornecido no trabalho, porém com um JARL antes da SUB que coloca o PC no endereço de outro JALR, que utiliza o registrador que foi salvo o PC no primeiro JARL para retornar e continuar a função a partir do SUB. Logo, testando tanto o pulo que o JALR deve fazer quanto o armazenamento do valor do PC antes do pulo.

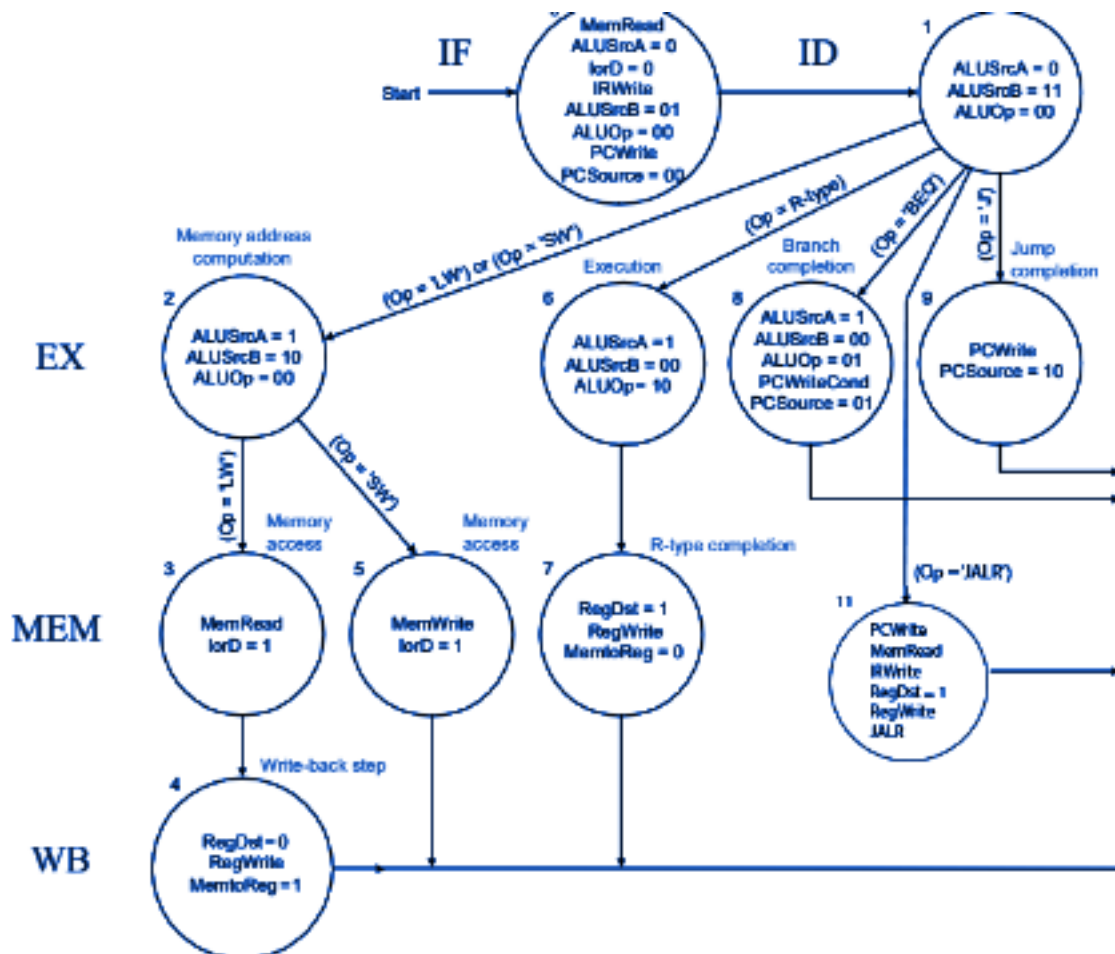
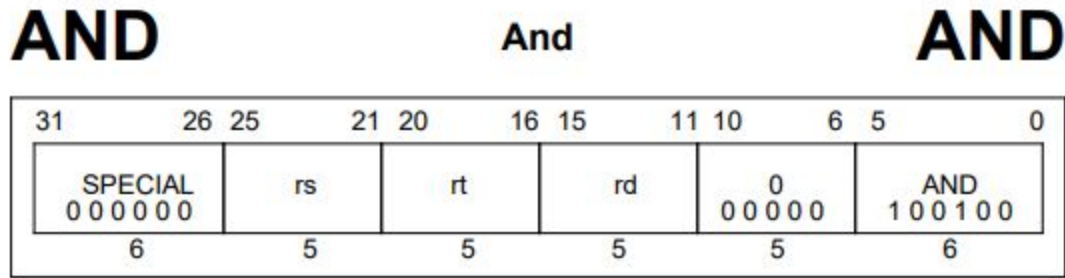


Figura 1: Máquina de estados multiciclo instrução JARL e AND.

AND rs,rt,rd



AND rs,rt,rd (opera rs AND rt e guarda o resultado em rd)

A instrução é interpretada da seguinte forma: Os bits *special* fazem com que a ROM de controle ative os seguintes bits de controle:

RegDst, ALUOp1 (que é o AND), RegWrite.

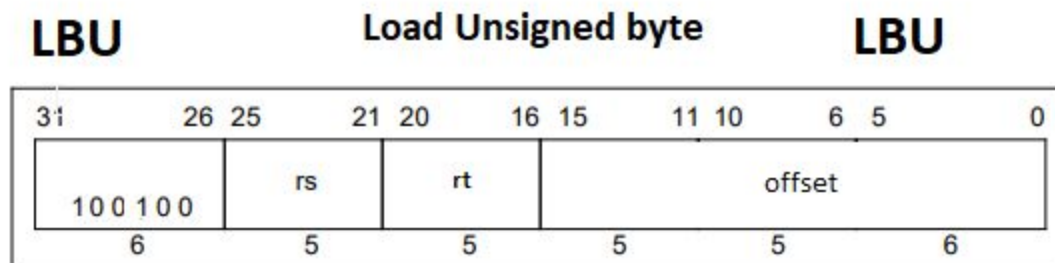
Os últimos 5 bits são redirecionados para o controlador da ULA (ALU Control) que faz com que a operação da ULA selecionada seja o AND lógico entre os dois operandos. O datapath deve direcionar os registradores de propósito geral *rs* e *rt* para a ULA que fará o AND e retornará para o registrador de escrita *rd*.

Mudanças:

Para os três datapaths, a instrução já estava implementada.

A máquina de estados do multiciclo segue o caminho de uma instrução do tipo R.

LBU rt, offset(base)



Para implementar o **LBU** em Monociclo, foi necessário adicionar uma porta **AND** com um elemento de valor constante 0xffffffc em uma de suas entradas. Na outra entrada é colocado o valor do endereço de memória. Ao realizar a operação AND desses dois valores, obtém-se o valor do endereço de memória com seus dois últimos bits zerados. A seguir, foi adicionado um MUX na saída dessa porta lógica, para selecionar o valor do endereço de memória original ou o endereço que passou pela porta **AND** e teve seus últimos bits zerados. A saída desse MUX envia os dados para o Data Memory.

A seguir, as saídas do Data Memory são divididas em 4 linhas de 8 bits e são conectadas a um novo MUX, que possui 4 entradas de 1 byte e 2 sinais de controle, que selecionará entre 1 dos bytes de saída do Data Memory. Os sinais de controle desse MUX são os dois últimos bits do endereço de memória. A saída desse MUX, que é um byte do Data Memory, é conduzida a um extensor de bits com tipo de extensão zero, que torna os bits de entrada compatíveis com a extensão de 32 bits. A saída do extensor é enviada para um MUX de 32bits, com 2 entradas, que seleciona entre os dados do Data Memory originais ou os dados estendidos. Esses dados estendidos são enviados ao registrador.

O Monociclo recebeu um bit de controle a mais chamado **LBU** que foi acionado e os bits de controle RegDst, MemRead, MemtoReg, ALUSrc, RegWrite também foram acionados para que a instrução ocorra corretamente.

No Multiciclo foram feitas as mesmas modificações que no monociclo com a adaptação de existir apenas uma memória para instruções e para dados. Os estados ligados são 1->c->d->4->0.

No Pipeline além das alterações feitas no monociclo foi adicionado um registrador em MEM/WB para guardar os dois últimos bits de memória da instrução. Também foi adicionado registradores em ID/EX, Ex/MEM e MEM/WB para guardar o bit de controle LBU.

As três arquiteturas de processador foram testadas antes e depois da implementação da instrução LBU. Os resultados são condizentes com o esperado.

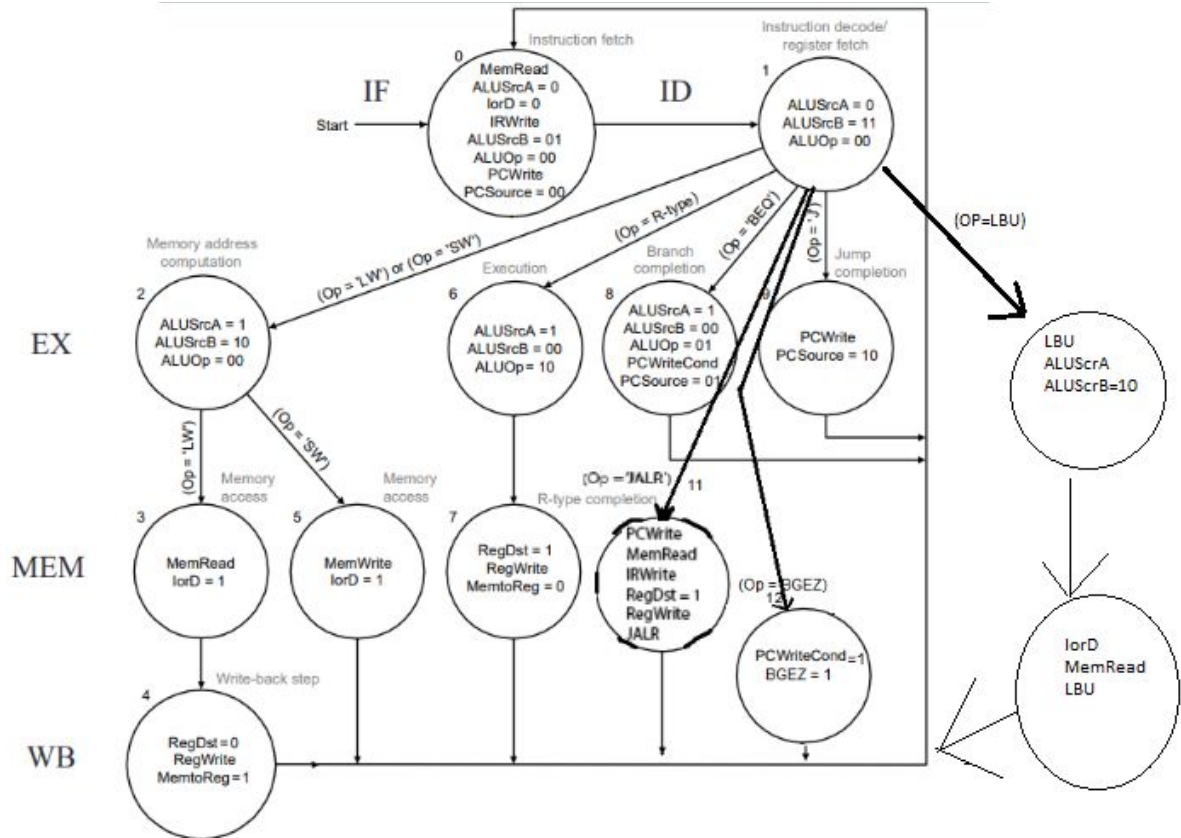
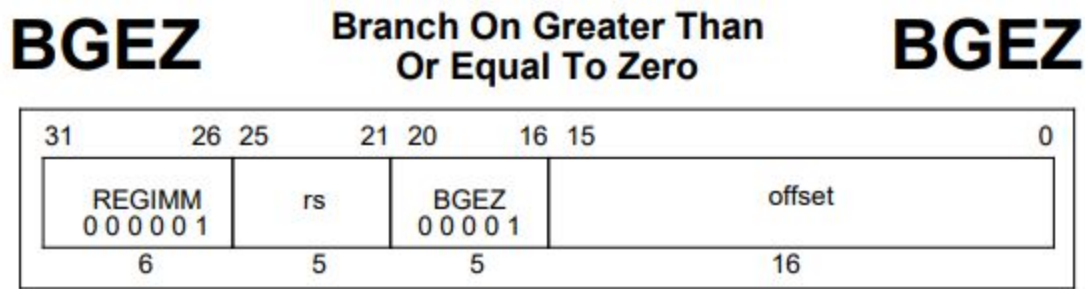


Figura 2: Máquina de estados multiciclo instrução LBU.

BGEZ R1, offset



Para todos os tipos de MIPS foi criado um bit de controle a mais chamado **BGEZ**

Monociclo:

Para poder implementar a instrução **BGEZ** primeiramente se fez um teste para verificar se o **REGMIMM** equivale ao 000001 e se também é o código 00001, referente ao **BGEZ**, nos bits 20 16. Caso ambos forem verdadeiros, via um **AND**, o bloco de controle será avisado de que efetivamente se recebeu a operação BGEZ e que deveremos efetuar o salto do *offset* caso o valor que está no registrador de propósito geral *rs* seja maior ou igual a zero ($rs \geq 0$).

O bloco de controle ligará então o bit de controle **BGEZ** e se testará se o valor no registrador RS passado na instrução é positivo ou zero. Para isso, se testa se o bit mais significativo do valor em RS é zero (se for zero, por ser complemento de 2 se garante ser positivo ou zero.)

Então se faz um **AND** junto com o bit de controle **BGEZ** para garantir que o branch vai ser feito e então somar ao PC o offset devidamente estendido o seu sinal e com 2 shift left.

Multiciclo:

No multicilo também se mantém a mesma ideia e funcionamento de verificar se a instrução é de fato **BGEZ** e o controle mandará para o estado correspondente da instrução (mais informação na imagem a seguir). Os bits de controle **BGEZ** e **PCWriteCond** serão ligados. O teste para verificar se o número é positivo ou maior que zero e então somar, ou não, o offset ao PC é feito da mesma forma que no monociclo (se bit mais significativo = 0, **AND BGEZ** está ligado **AND PCWriteCond** está ligado se soma o offset ao PC).

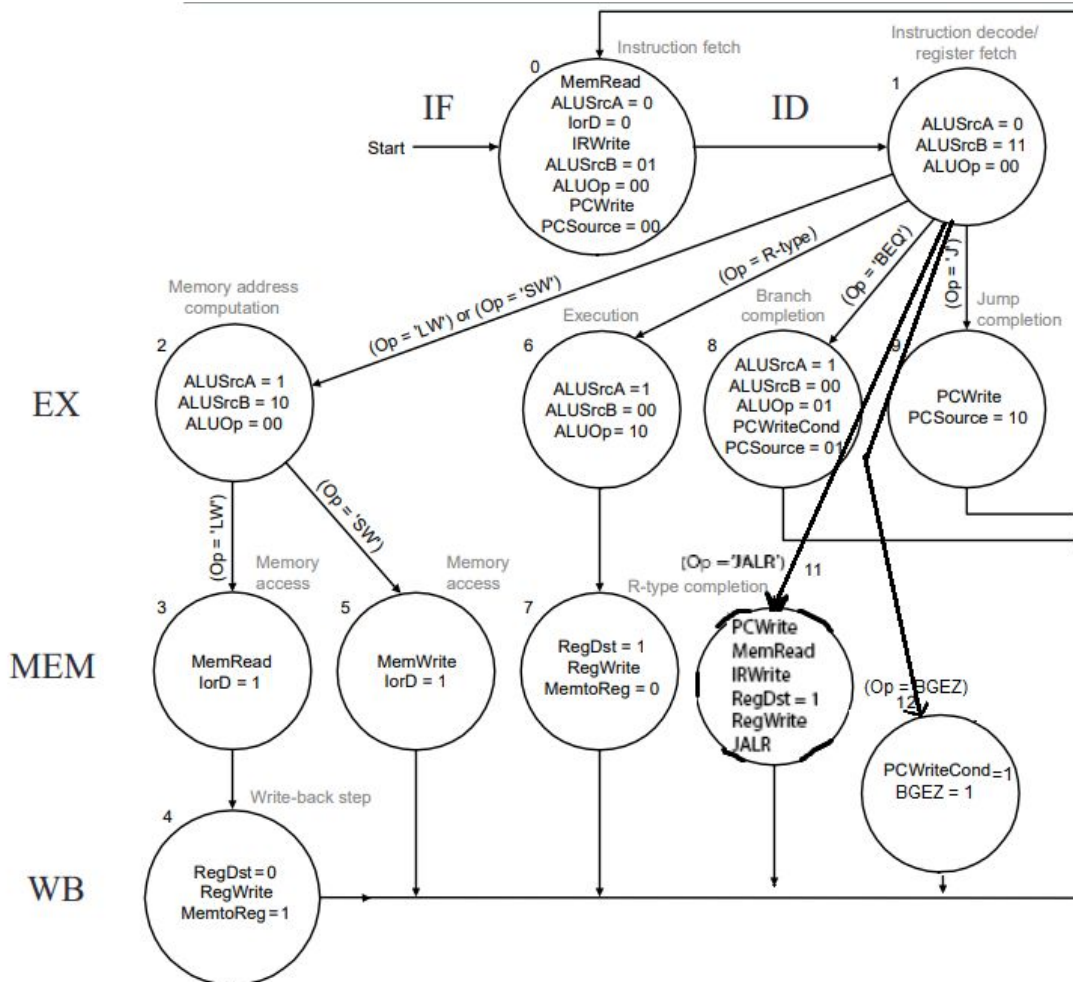


Figura 3: Máquina de estados multiciclo instrução BEGZ.

Pipeline;

No Pipeline se fez o mesmo teste para verificar se a instrução **BGEZ** ao igual que no mono e multiciclo. Então se avisa o controle e ele ligará apenas o bit de controle **BGEZ** quando a instrução estiver no estágio de **MEM** do pipe. Nesse caso, o valor do registrador RS passado na instrução é testado se for ≥ 0 no estágio de **EX** do pipe (da mesma forma que no mono e multiciclo) e o resultado é mandado para um registrador adicionado por nós para “salvar” esse valor para o próximo estágio do pipe, já que em **MEM** se testará se for **BGEZ AND** esse resultado de ser ≥ 0 para decidir se o offset será somado ou não ao PC.

Testes

Os testes do conjunto de instruções foi feito a partir do programa exemplo deixado no Moodle. JARL,AND, BEGZ, LBU . O princípio desse programa é replicar o que está no primeiro endereço da memória de dados para seguintes 5 endereços (a partir do terceiro). A Tabela 1 mostra como a memória de dados é inicializada.

Ele carrega as duas primeiras posições da memória de dados (no multiciclo, carrega os endereços 80 e 84) em R1 e R2, soma esses registradores e armazena em R3, faz a operação de JARL para outro JARL que por sua vez posiciona o PC na próxima instrução. Em seguida, subtrai R3 de R2 e armazena em R3 e compara R3 com R1. Se os registradores forem iguais (deu tudo certo), pula para LB3 e armazena R3 no endereço 8 (48 no multiciclo), se não, armazena R0 em 8. Caso a primeira etapa seja concluída com sucesso, o programa fará AND entre os registradores R1 e R2 e armazenará o resultado em R5, e salva o R5 na quarta posição de memória. Em seguida, testa se $R5 \geq 0$, caso positivo (deu certo) pula para LB2 onde é armazenado o R5 na quinta posição de memória. Carrega em R5 o valor na oitava posição da memória (número negativo) e testa se $R5 \geq 0$, (não pula - deu certo) armazena na sexta posição da memória o R5. Carrega em R3 os últimos 8 bits da nona posição memória e grava o valor na sétima posição da memória de dados.

Posição da memória	Conteúdo carregado (hex)	Significado do conteúdo
0	5	Operando que vai para o r1
1	A	Operando que vai para o r2
8	80000000	Operador que vai para o r5 em BGEZ (<0)
9	FFFFFF05	Operador que vai para o r5 para o teste do LBU
10	1D	Valor para deslocamento no PIPE

Tabela 1: conteúdo inicial da memória de dados.

A Tabela 2 apresenta os valores esperados pelas posições de memória depois de o teste ser executado.

Posição da memória	Valor esperado	Insts. testadas
3	5	LW, BEQ, JARL , ADD, SUB, SW, J
4	5	AND , SW
5	5	BGEZ (\geq), SW
6	5	BGEZ (\leq), SW
7	5	LBU , SW

Tabela 2: conteúdo esperado na memória.

Para todos os MIPS, os resultados esperados foram retornados.

Código do teste em assembly:

```
LW R1, 0(R0)
LW R2, 4(R0)
#Para o pipe, foi preciso: LW R7, 36(R0)
ADD R3, R1, R2
JARL R4, R2    # #Para o pipe, foi preciso: JARL R4, R7
SUB R3, R3, R2
BEQ R3, R1 LB3
SW R0, 8(R0)
J 0
SW R0, 8(R0)
J 0
LB1:
JARL R4, R4
LB3:
```

SW R3, 8(R0)
AND R3,R1,R5
SW R5,12(R0)
BEGZ R5, LB2
ADD R3, R1, R2
ADD R3, R1, R2
SW R3,16(R0)
LB2:
SW R5,16(R0)
LW R5, 28(R0)
BEGZ R5, LB4
SW R3, 20(R0)
BEQ R0, LB5
LB4:
SW R0,20(R0)
LB5:
LBU R3, 32(R0)
SW R3, 24(R0)

Em hexadecimal:

Monociclo

Memória de instrução:

```
8c010000 8c020004 00221820 00402009 00621822 10610005 AC000008 08000000
ac030008 08000000 00804009 ac030008 00232824 AC05000c 04A10003 00221820
00221820 ac030010 ac050010 8c05001C 04A10002 ac030014 04010001 ac000014
90030020 ac030018
```

Memória de dados:

```
5 a 5*0 80000000 ffffff05
```

Multiciclo

Memória Principal:

```
8c010080 8c020084 00221820 00402009 00621822 10610005 8c020088 08000000
ac030048 08000000 00804009 ac030088 00232824 AC05008C 04A10003 00221820
00221820 ac030090 ac050090 8c05009C 04A10002 ac030094 04010001 ac000094
900300A0 ac030098 6*0 5 a 5*0 80000000 ffffff05
```

Pipeline

Memória de instrução:

```
v2.0 raw 8c010000 8c020004 8c070024 0 0 0 00221820 00E02009 0 0 0 00621822 0 0
0 10610011 0 0 0 AC000008 08000000 0 0 0 ac030008 08000000 0 0 0 00804009 0 0
0 ac030008 00232824 0 0 0 AC05000c 04A1000B 0 0 0 00221820 0 0 0 00221820 0 0
0 ac030010 ac050010 8c05001C 0 0 0 04A10008 0 0 0 ac030014 04010004 0 0 0
ac000014 90030020 0 0 0 ac030018
```

Memória de dados:

```
v2.0 raw 5 a 5*0 80000000 ffffff05 1D
```