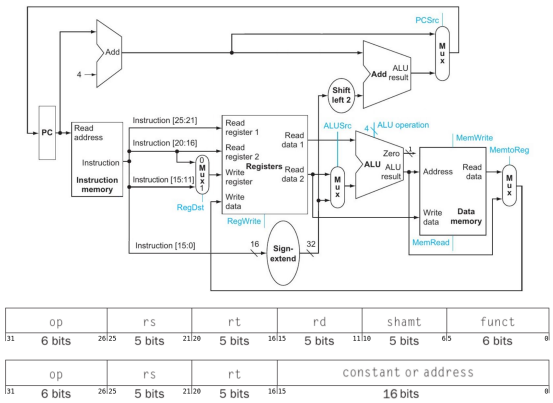
Sinais de Controle

Yuri Kaszubowski Lopes

UDESC

Anotações

Revisão: Caminho de dados



Anotações

Controle da ALU

- Temos uma unidade de controle exclusiva da ALU (e outra geral)
- Sinal **ALU Operation** de 4 bits
- O sinal enviado para a ALU vai depender da instrução, e.g.:
  - `lw` e `sw` precisam calcular o endereço através de uma adição: 0010_2
  - `beq` precisa realizar uma subtração (0110_2) para verificar se os valores são iguais
  - Instruções do **tipo-R** devem ter a operação definida pelo campo `func` de 6 bits da instrução

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

Anotações

Controle da ALU

- A unidade de controle da ALU vai receber como entrada um sinal de 2 bits, chamado **ALUOp**, que vai definir o tipo da instrução, e também vai receber o sinal do campo `funct`
- **ALUOp**:
  - $00_2 \rightarrow$ indica que a operação é uma **adição** (para `lw` e `sw`)
  - $01_2 \rightarrow$ indica que a operação é uma **subtração** (para `beq`/`bne`)
  - $10_2 \rightarrow$ indica que a operação vai ser definida pelo campo `funct` (tipo-R)

Anotações

Controle da ALU: Tabela verdade

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

- **ALUOp** e *Funct field*: entradas do Controle da ALU
- *ALU control input* (**ALU Operation**): saída para a ALU

Anotações

Controle da ALU: Tabela verdade

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

- Os dois primeiros campos do `funct` não são relevantes na escolha da operação
- No entanto, todos os campos de `funct` serão enviados para o controle da ALU, pois uma implementação mais ampla do MIPS pode precisar desses campos.

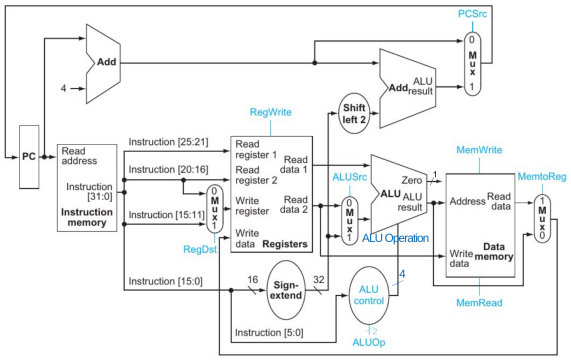
Anotações

Controle da ALU

- Dada a tabela verdade, podemos agora construir o Controle da ALU
- Podemos definir a expressão Booleana para a ALU, simplificá-la, e implementá-la com portas lógidas
- Exemplo: utilizar soma dos produtos, e então simplificar a expressão lógica pelos teoremas e leis booleanas

Anotações

Controle da ALU: Tabela verdade



- O controle da ALU gera o sinal **ALU Operation** de 4 bits
- Mas depende de um sinal **ALUOp** de 2 bits

Anotações

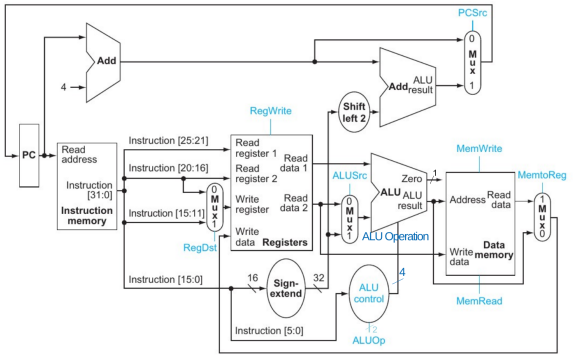
Controle da ALU

- O sinal **ALUOp** vai ser gerado pela unidade de controle principal
  - Múltiplos níveis de unidades de controle
  - Mais simples projetar
  - Possível redução no tamanho do circuito
  - Possível aumento de velocidade
  - Unidades mais simples processam a informação mais rapidamente do que uma única unidade grande
  - Redução do período do ciclo de clock (principalmente quando considerarmos pipelining

Anotações

Unidade de Controle Principal

- Podemos agora criar uma unidade de controle principal
  - Vai cuidar das 7 linhas de controle (de 1 bit) e do sinal de controle de dois bits **ALUOp**



Anotações

Exercício

- Verifique os sinais no circuito, e escreva na tabela o que se espera quando cada um dos sinais é 0 ou 1. O resultato em RegDST está descrito como exemplo.

Sinal	Efeito esperado quanto 0	Efeito esperado quanto 1
RegDst	O núm. do reg. de destino deve vir do campo rt (bits [20:16]).	O núm. do reg. de destino deve vir do campo rd (bits [15:11])
RegWrite		
ALUSrc		
PCSrc		
MemRead		
MemWrite		
MemtoReg		

Anotações

Exercício

- Verifique os sinais no circuito, e escreva na tabela o que se espera quando cada um dos sinais é 0 ou 1. O resultato em RegDST está descrito como exemplo.

Sinal	Efeito esperado quanto 0	Efeito esperado quanto 1
RegDst	O núm. do reg. de destino deve vir do campo rt (bits [20:16]).	O núm. do reg. de destino deve vir do campo rd (bits [15:11])
RegWrite	Nada acontece	Os dados enviados ao banco de registradores são escritos no endereço do registrador de escrita.
ALUSrc	O segundo operando da ALU vêm do segundo registrador fonte lido do banco de registradores.	O segundo operando da ALU vêm do campo imediato da instrução (16 bits estendidos para 32).
PCSrc	PC recebe PC+4	PC recebe o endereço de desvio calculado
MemRead	Nada acontece	O conteúdo da memória no endereço especificado é enviado para a saída de leitura da memória
MemWrite	Nada acontece	O conteúdo da memória no endereço especificado é substituído pelo dado na entrada "WriteData" da memória
MemtoReg	O valor a ser escrito no registrador de destino deve vir da ALU	O valor a ser escrito no registrador de destino deve vir da Memória

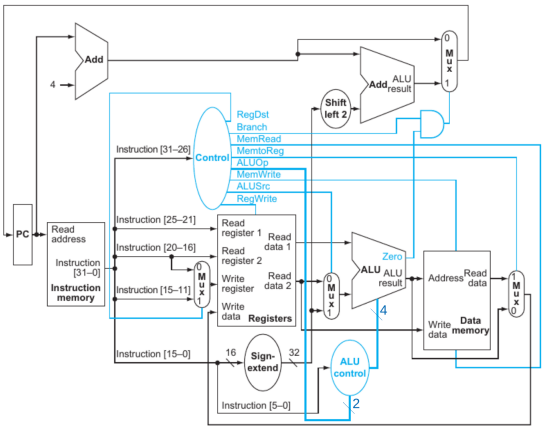
Anotações

Tabela verdade da Unidade de Controle

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Anotações

Unidade de Controle



Anotações

Controle Hardwired

- Nossa unidade de controle é simples e pode ser do tipo “hardwired”
- Definida com portas lógicas e de comportamento fixo

Anotações

Microcódigo

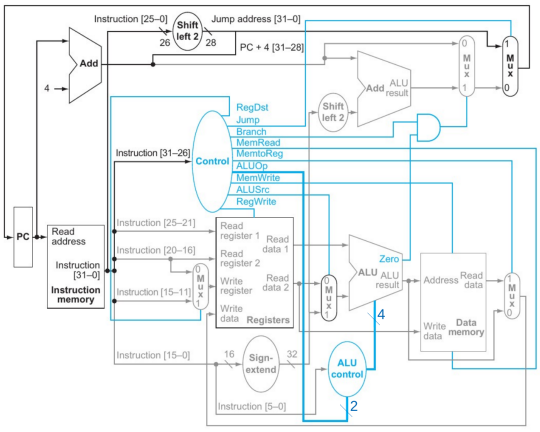
- Em projetos complexos podemos criar unidades de controle programáveis
- O programa na unidade de controle dita como os sinais de controle são gerados de acordo com as entradas: O programa é chamado de microcódigo
- Utilizado na maioria das CPUs
- Temos uma maior flexibilidade
- Podemos realizar correções no hardware “on-the-fly”
- Assim que a Intel “corrigiu” as falhas de segurança Spectre e Meltdown em suas CPUs sem precisar trocá-las
- No Linux, abra um terminal e digite `dmesg | grep` microcode para verificar sua versão de microcódigo na CPU
- Muitas vezes o microcódigo é chamado de *firmware*
- O programa que controla o fluxo interno do hardware
- Não cobriremos os detalhes sobre microcódigo em nossa disciplina introdutória

Anotações

Exercícios

1. Adicione o circuito para realizar jumps em nosso processador. Ligue os sinais de controle, e se necessário crie novos, indicando na tabela verdade quais as entradas e saídas necessárias para os sinais de controle dessa instrução.
2. Considerando as instruções implementadas até o momento, qual a instrução que você considera que demora mais tempo e qual demora menos tempo para ser executada? Uma adição? Loads? Stores? beq? jump? ... Explique
3. Adicionando mais um sinal de controle, `BranchNE`, e portas lógicas, indique as alterações no caminho de dados para a implementação da instrução `bne`

Anotações



Anotações

Referências

- D. Patterson; J. Henessy. **Organização e Projeto de Computadores: Interface Hardware/Software**. 5a Edição. Elsevier Brasil, 2017.
- Andrew S. Tanenbaum. **Organização estruturada de computadores**. 5. ed. São Paulo: Pearson, 2007.
- Harris, D. and Harris, S. **Digital Design and Computer Architecture**. 2a ed. 2012.
- courses.missouristate.edu/KenVollmar/mars/

Anotações

Anotações

Anotações