

TRABALHO PRÁTICO 2018/2

EXTENSÃO DE INSTRUÇÃO DE PROCESSADOR ARM MONOCÍCLO.

Nota máxima: 10 pontos corresponde a 100% da nota.

Prazos:

- Entrega no Moodle: **13/12/2018**
- Apresentação ao Professor: até dia 14/12/2018.
 - É de responsabilidade da dupla o agendamento de horário junto ao professor para demonstração da implementação, que deverá ser fora do horário de aula.
 - Avaliação da implementação (nota do grupo – 50%) e avaliação da entrevista (nota individual – 50%)
 - Trabalho, no máximo, em grupos de 3 participantes.

Especificação

Você estenderá um processador simplificado de ciclo único ARM usando o SystemVerilog. Um programa de teste deverá ser carregado na memória e confirmará se o sistema funciona. Em seguida, você deve implementar quatro novas instruções e, em seguida, escrever um adicionar estas instruções no programa de teste, para que se confirme que as novas instruções também funcionam. O esquema do processador de ciclo único é apresentado no final para sua conveniência. Essa versão do processador de ciclo único ARM e, neste estágio, pode executar as seguintes instruções: ADD, SUB, AND, ORR, LDR, STR e B.

Processador ARM Single-Cycle

O módulo ARM single-cycle em SystemVerilog (arm_single.sv) pode ser encontrado no repositório: [1]

Estude o arquivo até estar familiarizado com seu conteúdo. Abra o arquivo arm_single.sv. O módulo de nível superior (chamado top) contém o processador arm (arm) e as memórias de dados e de instruções (dmem e imem). Agora olhe para o módulo do processador (chamado arm), ele instância dois sub-módulos, “controller” e “datapath”. Agora, observe o módulo controller e seus submódulos. Ele contém dois sub-módulos: “decode” e “condlogic”. O módulo decode produz todos menos três sinais de controle. O módulo condlogic produz os três sinais de controle restantes que atualizam o estado arquitetural (RegWrite, MemWrite) ou determinam o próximo PC (PCSrc). Estes três sinais dependem da condição da instrução (Cond3:0) e dos sinalizadores (flags) de condições armazenadas (Flags3:0) que são internos ao módulo condlogic. As flags de condição produzidas pela ALU (ALUFlags3:0) são atualizadas nos registros de flags dependentes do bit S (FlagW1:0) e se a instrução é executada (novamente, dependendo da condição Cond3:0 e do valor armazenado das flags de condição Flags3:0).

A instrução e as memórias de dados instanciadas no módulo “top” são, cada uma, uma matriz de 64-palavras \times 32 bits. A memória da instrução precisa conter alguns valores iniciais representando o programa. O programa de teste é dado no arquivo “memfile.s” do repositório apresentado anteriormente. Estude o programa até entender o que ele faz. O código já em idioma de máquina para o programa pode ser encontrado com o nome de “memfile.dat” no repositório.

Parte 1 (3 pts) – Testando o processador ARM monociclo.

Em um sistema complexo, se você não sabe o que esperar da resposta, é improvável que você receba a resposta certa. Comece prevendo o que deve acontecer em cada ciclo ao executar o programa.

1 - Desenhe o esquemático correspondente ao HDL em SystemVerilog do processador ARM. Ilustre a hierarquia do testbench e os módulos internos ao DUT (Device Under Test). Apresente no esquemático as estruturas de conexão de forma simplificada (para fios em paralelo, trace apenas 1 fio e ilustre o número de fios que compoe tal barramento simples).

2 - Preencha o gráfico na Tabela 1 no final com suas previsões da execução do código memfile.s do repositório fornecido. Que endereço escreverá a instrução final do STR e qual valor ela escreverá? Simule seu processador com o ModelSim. Adicione todos os sinais da Tabela 1 à sua janela de ondas. Execute a simulação. Se tudo correr bem, o testbench imprimirá “Simulação bem-sucedida”. Observe as formas de onda e verifique se elas correspondem às suas previsões na Tabela 1.

Parte 2 (7 pts): - Modificando o processador ARM monociclo

1 - Estenda as instruções em nível de microarquitetura do processador ARM monociclo, com código HDL disponível em [1], de forma a habilitá-lo a processar as instruções **TST, CMP, LSL, MOV, EOR e LDRB**.

2 - Para validar, estenda o código de testes utilizado na parte , gere o novo código memfile2.s e memfile2.dat, com os códigos correspondentes de montagem e de máquina. Por fim, atualize a tabela 1 com os dados obtidos dessa nova simulação com as novas instruções suportadas.

Apresentação [10pts]

Na apresentação, deverá ser entregue o relatório final com a apresentação do trabalho realizado. Além disso, será necessária a demonstração da execução do trabalho no ModelSim junto ao professor, com avaliação individual.

A nota final do trabalho será a média entre o somatório das partes 1 e 2 com a nota da apresentação (relatório + entrevista).

Pontuação Bônus (extra)

- Trabalhos entregues e apresentados, funcionando (parte 1 e 2), antes do dia 03/12/18: 1 (um) ponto na média.
- Trabalhos entregues com código versionado em git em repositório público: 2,00 (dois) pontos na nota do trabalho (versionado será considerado o código com no mínimo 5 commits que mostrem e registrem a evolução do trabalho em cada etapa do desenvolvimento, maiores informações em [4]). Não faça a edição online do código, isso atrapalhará o desenvolvimento. O versionamento correto é realizado com trabalho independente de cada membro da equipe.

Referências

- [1] <https://gitlab.com/arqcomp/ARMProcessors>
- [4] <https://tableless.com.br/iniciando-no-git-parte-1/>

ESQUEMÁTICO DO PROCESSADOR ARM MONOCÍCLO DE REFERÊNCIA

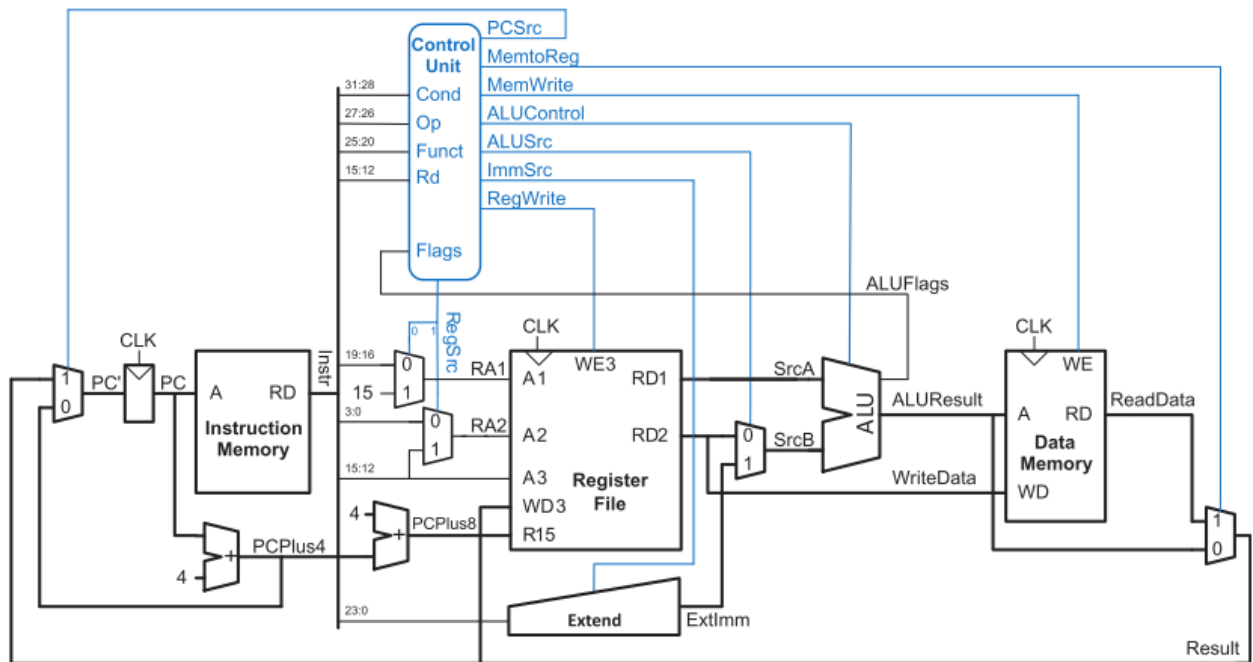


Figure 7.13 Complete single-cycle processor

INSTRUÇÕES BÁSICAS DE

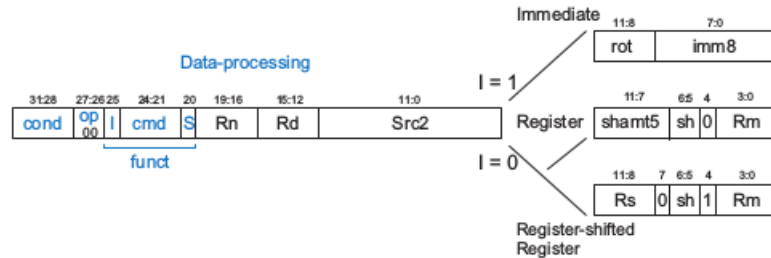


Figure B.1 Data-processing instruction encodings

Table B.1 Data-processing instructions

cmd	Name	Description	Operation
0000	AND Rd, Rn, Src2	Bitwise AND	$Rd \leftarrow Rn \& Src2$
0001	EOR Rd, Rn, Src2	Bitwise XOR	$Rd \leftarrow Rn \wedge Src2$
0010	SUB Rd, Rn, Src2	Subtract	$Rd \leftarrow Rn - Src2$
0011	RSB Rd, Rn, Src2	Reverse Subtract	$Rd \leftarrow Src2 - Rn$
0100	ADD Rd, Rn, Src2	Add	$Rd \leftarrow Rn + Src2$
0101	ADC Rd, Rn, Src2	Add with Carry	$Rd \leftarrow Rn + Src2 + C$
0110	SBC Rd, Rn, Src2	Subtract with Carry	$Rd \leftarrow Rn - Src2 - \bar{C}$
0111	RSC Rd, Rn, Src2	Reverse Sub w/ Carry	$Rd \leftarrow Src2 - Rn - \bar{C}$
1000 ($S = 1$)	TST Rd, Rn, Src2	Test	Set flags based on $Rn \& Src2$
1001 ($S = 1$)	TEQ Rd, Rn, Src2	Test Equivalence	Set flags based on $Rn \wedge Src2$
1010 ($S = 1$)	CMP Rn, Src2	Compare	Set flags based on $Rn - Src2$
1011 ($S = 1$)	CMN Rn, Src2	Compare Negative	Set flags based on $Rn + Src2$
1100	ORR Rd, Rn, Src2	Bitwise OR	$Rd \leftarrow Rn Src2$
1101	Shifts:		
$I = 1$ OR ($instr_{11:4} = 0$)	MOV Rd, Src2	Move	$Rd \leftarrow Src2$
$I = 0$ AND ($sh = 00$; $instr_{11:4} \neq 0$)	LSL Rd, Rm, Rs/shamt5	Logical Shift Left	$Rd \leftarrow Rm \ll Src2$
$I = 0$ AND ($sh = 01$)	LSR Rd, Rm, Rs/shamt5	Logical Shift Right	$Rd \leftarrow Rm \gg Src2$

PROCESSAMENTO DE DADOS

ANEXO 1 – TABELA 1.[illegible]