Implementação do Processador ARM7 em ArchC

Wesley Nunes Gonçalves

23 de novembro de 2007

ARM7

Instruções Implementadas

O ARM possui 37 registradores, sendo 31 registradores de propósito geral e 6 registradores de *status*. Dos 31 registradores de propósito geral, 16 são visíveis e 15 são utilizados para acelerar o processamento de exceções. Neste trabalho, estão implementados os 16 registradores visíveis, incluindo o registrador PC. Dos registradores de *status*, o registrador *Current Program Status Register* (CPSR) foi implementado para armazenar as *flags* de controle. As *flags* são geralmente modificadas por instruções de comparação (CMN, CMP, TEQ e TST) e execução de instruções aritméticas.

As instruções do processador ARM7 são divididas em diversos tipos, neste trabalho foram implementadas as instruções dos tipos de multiplicação (*Multiply*), processamento de dados (*Data-processing*) e desvio (*Branch*) visualizadas nas Tabelas 1, 2 e 3, respectivamente. Além disso, foi implementado o modo de endereçamento utilizando um registrador ou um imediato e o formato das instruções de acesso a memória foram definidas. Para a execução dessas instruções, principalmente das instruções do tipo *Branch*, foi necessário implementar a decodificação do campo COND, presente em todas instruções. Essa campo permite que uma instrução seja executada condicionamente de acordo com as flags, estado do processador, entre outros. Por exemplo, uma instrução B com o campo COND=0x00 irá ser executada se a Flag Z=1. Para a execução das instruções foi implementado o processador com 3 estágios: Instruction Fetch - IF, Instruction Decode - ID e Execute - EX.

Mnemônico	**Ação**
MUL	Rd = Rm * Rs
MLA	Rd = Rm * Rs + Rn

Tabela 1: Instruções do tipo *Multiply*.

Mnemônico	**Ação**
AND	Rd = Rn AND shifter_operand
EOR	Rd = Rn EOR shifter_operand
SUB	Rd = Rn - shifter_operand
RSB	Rd = shifter_operand - Rn
ADD	Rd = Rn + shifter_operand
ADC	Rd = Rn + shifter_operand + Flag C
SBC	Rd = Rn - shifter_operand - !(Flag C)
RSC	Rd = shifter_operand - Rn - !(Flag C)
TST	Atualiza Flags após Rn AND shifter_operand
TEQ	Atualiza Flags após Rn EOR shifter_operand
CMP	Atualiza Flags após Rn - shifter_operand
CMN	Atualiza Flags após Rn + shifter_operand
ORR	Rd = Rn OR shifter_operand
MOV	Rd = shifter_operand
BIC	Rd = Rn AND !(shifter_operand)
MVN	Rd = !(shifter_operand)

Tabela 2: Instruções do tipo *Data-processing*.

Mnemônico	**Ação**
B	Jump signed_immed
BL	LR = PC, Jump signed_immed, ..., MOV PC, LR

Tabela 3: Instruções do tipo *Branch*.

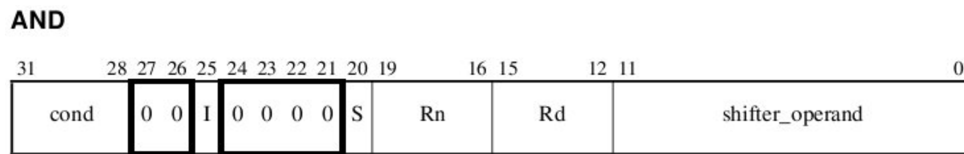

Figura 1: Bits da Instrução AND.

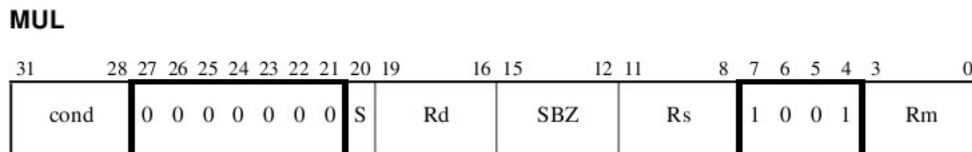

Figura 2: Bits da Instrução MUL.

Dificuldades na Implementação

A principal dificuldade encontrada foi entender a arquitetura do ARM7 e a linguagem ArchC para iniciar a implementação. A grande dificuldade na arquitetura do ARM7 foi com relação as *Flags*, o campo COND e como isso influencia a execução das instruções do tipo *Branch*. Além disso, uma dificuldade encontrada foi com campos iguais nas instruções AND e MUL. Para identificar se a instrução é uma AND ou MUL é necessário verificar os campos em negrito ilustrados nas Figuras 1 e 2 respectivamente. Como pode ser visto, pode ocorrer a decodificação de uma instrução MUL como se fosse uma instrução AND.

Para resolver esse problema, foi necessário declarar a instrução MUL antes da instrução AND, como pode ser visto no trecho de código abaixo. Com isso, a instrução é sempre avaliada antes como MUL e depois como AND.

```
ac_instr<Multiply> mla, mul;
ac_instr<Multiply_Long> umull, umlal, smull, smlal;
ac_instr<Data_Processing> and, add, adc
ac_instr<Branch> b, bl;
ac_instr<Branch_and_Exchange> bx, blx;;
```

Sintaxe de Programas

A ferramente segue a mesma sintaxe estabelecida pela arquitetura ARM7. Como exemplo, vamos utilizar um programa para calcular o fatorial abaixo:

```
int contador = 2;
```

```
int fatorial = 1;
int n = 5;

fat: fatorial = fatorial * contador;
     contador = contador + 1;
     if(contador <= n) GOTO fat;
```

Um possível código, com instruções do ARM7 é mostrado abaixo, com r1 = contador, r2 = fatorial e r3 = n:

```
1) ADD r1, $zero, 2
2) ADD r2, $zero, 1
3) ADD r3, $zero, 5
4) MUL r2, r2, r1
5) ADD r1, r1, 1
6) CMP r1, r3
7) B -5
8) NOP
9) NOP
```

A instrução B recebe como *label* o número de instruções que deve ser somada ao PC atual, levando em consideração os estágio do pipeline. Quando a instrução B for executada, o PC estará apontando para a instrução 9 (segundo NOP) devido os estágios do pipeline. Com isso, para retornarmos para a instrução 4, é necessário somarmos -5. Os dois NOPs após a instrução B são necessário para resolver o branch e o programa não seja finalizado com a chamado do *syscall* ou execute uma instrução que não seria executada.

As instruções em hexadecimal são apresentadas abaixo:

```
0000 E2901002
0004 E2902001
0008 E2903005
000C E0120291
0010 E2911001
0014 E1510003
0018 DAFFFFFB
001C F0000000
0020 F0000000
0024 0000000C
```

É importante notar o campo COND de cada instrução (primeiro valor: **E**2911001), por exemplo, se COND=E a instrução sempre será executada, se COND=D a instrução será executada se as *flags* Z == 1 ou N != V (flags setadas pela instrução CMP - endereço 0014, representando r1 ¡= r3), se COND=F a instrução não será executada.

Conclusão e Trabalhos Futuros

Neste trabalho foi implementada uma versão simplificada do processador ARM7. Com essa implementação é possível executar programas que usam instruções aritméticas, de desvio condicional (if e loop) e incondicional. O processador executa as instruções através de 3 estágios de pipeline. Para o controle, algumas flags do registrador de *status* são setadas após a execução de algumas instruções.

Como trabalhos futuros estão a implementação do comportamento das instruções de acesso a memória e definição e implementação de outras instruções, como as instruções do coprocessador. Além disso, seria interessante implementar os outros modos de endereçamento das instruções de processamento de dados.