
**Universidade Católica Dom Bosco**
Curso de Bacharelado em Engenharia de Computação

# Relatório da Implementação de um Simulador para o Processador ARM740T

Fábio Alencar Lima

Profª. Dr. Ricardo Ribeiro dos Santos

UCDB - Campo Grande - MS - 11/2007

# Resumo

Este relatório é referente ao desenvolvimento de um simulador para o processador ARM740T com o auxílio da ferramenta ArchC. Com essa ferramenta foi possível descrever alguns comportamentos de instruções existentes nesse processador, porém muitas outras foram deixadas de lado. As próximas sessões estarão abordando sobre o que foi feito durante o desenvolvimento desse trabalho, bem como as dificuldades encontradas.

# Abstract

This report is for the development of a simulator for the processor ARM740T with the aid of the tool ArchC. The tool was possible to describe some behavior of this processor instructions, but many others have been left aside. The next sessions will be addressing on what has been done during the development of this work, and the difficulties encountered.

# Conteúdo

# Capítulo 1

# Introdução

Desde que o primeiro processador foi criado até o presente momento, ele sofreu muitos avanços, e com certeza sofrerá muitos mais. No entanto, para isso, novas técnicas e ferramentas para o desenvolvimento de processadores foram criadas, permitindo assim que um processador seja desenvolvido logicamente e posteriormente simulado em software para alterações alterações venham ser feitas antes mesmo que ele seja criado fisicamente.

O ArchC é uma linguagem de alto nível desenvolvida pela UNICAMP para descrever todo o comportamento de um processador e também os ciclos necessários para que cada instrução seja concluída. Com o auxílio dessa linguagem e mais uma outra linguagem, chamada SystemC, é possível criar um simulador para sistemas embarcados ou não. Desse modo foi escolhida essa linguagem para criar um simulador de um processador baseado na arquitetura RISC, chamado ARM74T.

O ARM740T é um processador com a capacidade de processar tanto instruções de 32 bits quanto instruções de 16 bits, chamada Thumb. Ele carrega boa parte das características empregadas no primeiro ARM, inclusive o pipeline de 3 estágios.O primeiro processador ARM foi desenvolvido entre os anos de 1983 e 1985 pela Acorn Computers Limited of Cambridge. Mais tarde, em 1990, para explorar melhor a tecnologia ARM ela foi separada formando então a ACORN Limited. Esse processador foi desenvolvido com base na arquitetura RISC visando um menor consumo de energia e baixo custo, para ser empregado em sistemas embarcados. O Capítulo 2 deste relatório possui quatro sessões estarão abordando o que foi implementado no para esse trabalho, o que faltou implementar para esse trabalho, as dificuldades encontradas para o desenvolvimento do trabalho, e por último, a conclusão desse trabalho.

# Capítulo 2

# O que foi Implementado

Para esse trabalho, foi feita a descrição do comportamento do ARM740T com o auxílio da linguagem AchC. Essa descrição variou desde a forma em que o processador trata as instruções até a forma em que o pipeline as executa. A primeira etapa foi criar um arquivo para descrever as instruções base para processamento de dados (DPI), também foi desrita a instrução base de multiplicação (MULT), e por último foi descrita a instrução base de salto (BRANCH), Figura 1.1 (a). Além das instruções base, foram descritas ainda as instruções de adição e multiplicação, pertencentes as instruções de DPI e MULT, com a definição do nome para elas e também de como são decodificadas, Figura 1.1 (b).

Após essa descrição, foi criado um outro arquivo para descrever os registradores de *pipeline*, o tamanho de palavra e o tamanho de memória, o número de registradores, os estágios do pipeline, o registrador de status, e o formato de leitura da instrução na memória, Figura 1.2.

Por último foi criado um outro arquivo contendo o comportamento das instruções dentro do *pipeline*, Figura 1.3.

Para que as instruções de processamento de dados pudessem ser decodificadas corretamente, foi necessário criar uma tabela contendo todas essas instruções juntamente com o código de cada uma em hexadecimal, Tabela 1.1. Além dessa codificação, foi criada também uma outra tabela contendo a codificação em hexadecimal do campo COND presente em todas as instruções do ARM740T, Tabela 1.2.

```
1  AC_ISA(arm740t){
2     ac_format Type_DPI = "%cond:4 %f:3 %opcode:4 %s:1 %rd:4 %rn:4 %operando:12";
3
4     ac_format Type_MULT = "%cond:4 %opcode:5 %u:1 %a:1 %s:1 %rd:4 %rn:4 %rs:4 %reservado:4 %rm:4";
5
6     ac_format Type_B = "%cond:4 %opcode:3 %l:1 %offset:24";
7
8     ac_instr<Type_DPI> = add;
9     ac_instr<Type_MULT> = mul;
10
11    ISA_CTOR(arm740t){
12        add.set_asm("add %rd %rn %rm");
13        add.set_decoder(cond=0x0E, f=0x00, opcode=0x01, s=0x01);
14
15        mul.set_asm("mul %rd %rn %rs");
16        mul.set_asm(cond=0x0E, opcode=0x01, u=1, a=0x00, s=0x01);
17    };
18 };
```

(*a*)

```
1  AC_ISA(arm740t){
2     ac_format Type_DPI = "%cond:4 %f:3 %opcode:4 %s:1 %rd:4 %rn:4 %operando:12";
3
4     ac_format Type_MULT = "%cond:4 %opcode:5 %u:1 %a:1 %s:1 %rd:4 %rn:4 %rs:4 %reservado:4 %rm:4";
5
6     ac_format Type_B = "%cond:4 %opcode:3 %l:1 %offset:24";
7
8     ac_instr<Type_DPI> = add;
9     ac_instr<Type_MULT> = mul;
10
11    ISA_CTOR(arm740t){
12        add.set_asm("add %rd %rn %rm");
13        add.set_decoder(cond=0x0E, f=0x00, opcode=0x01, s=0x01);
14
15        mul.set_asm("mul %rd %rn %rs");
16        mul.set_asm(cond=0x0E, opcode=0x01, u=1, a=0x00, s=0x01);
17    };
18 };
```

(*b*)

Figura 2.1: Representação do código para descrição das instruções de DPI, MULT, e BRANCH na Figura (a) marcado com retângulo, e na Figura (b) marcado com um retângulo, está o código para definir as instruções que fazem parte da de DPI e da de MULT e também como são decodificadas.

```
2  AC_ARCH(arm740t){
3      ac_wordsize 32;
4      ac_mem DM:32M;
5
6      ac_regbank RB:36;
7
8      ac_format F_CPSR = "%N:1 %Z:1 %C:1 %V:1 %reservado:20 %I:1 %F:1 %T:1 %mode:5";
9
10     ac_format F_IF_ID = "%npc:32";
11     ac_format F_ID_EX = "%npc:32 %wdata:32 %res:32 %imm:32 %data1:32 %data2:32
12                          %rd:4 %rn:4 %rm:4 %regwrite:1 %memread:1 %memwrite:1";
13
14     ac_reg<F_IF_ID> IF_ID;
15     ac_reg<F_ID_EX> ID_EX;
16
17     ac_reg<F_PSR> CPSR;
18
19     ac_stage   IF,ID,EX;
20
21
22     ARCH_CTOR(arm740t) {
23
24         ac_isa("arm740t_isa.ac");
25         set_endian("big");
26     };
27 };
```

Figura 2.2: Representação do código para descrição do pipeline, dos registradores, da memoria, do tamanho de palavra aceita, e do registrador de status.

```
11  void ac_behavior(instruction){
12
13    switch( stage ) {
14    case IF:
15      ac_pc += 4;
16      IF_ID.npc = ac_pc;
17      break;
18
19    case ID:
20      break;
21
22    case EX:
23
24      if (ID_EX.regwrite == 1) {
25
26        if (ID_EX.rdest != 0)
27          RB.write(ID_EX.res, ID_EX.wdata);
28      }
29      break;
30
31      default:
32      break;
33    }
34  };
35
```

(a)

```
36  void ac_behavior(Type_DPI){
37      switch(stage){
38          case IF:
39          break;
40
41          case ID:
42              ID_EX.npc = IF_ID.npc;
43              ID_EX.regwrite = 1;
44              ID_EX.memread = 0;
45              ID_EX.memwrite = 0;
46          break;
47
48          case EX:
49          break;
50
51          default:
52          break;
53      }
54  };
```

(b)

```
56  void ac_behavior(add){
57      switch(stage){
58      case IF:
59      break;
60
61      case ID:
62      break;
63
64      case EX:
65              ID_EX.res =ID_EX.data1+ID_EX.data2;
66          ID_EX.regwrite = 1;
67          fprintf("Valor da Soma:%d", ID_EX.res);
68      break;
69
70      default:
71          break;
72      }
73  };
```

(c)

Figura 2.3: Figura (a) Descreve o comportamento genérico das instruções no *pipeline*, a Figura (b) descreve o comprtamento das instruções de processamento de dados no pipeline, e a Figura (c) faz a descrição do comportamento da instrução add no *pipeline*.

| Mnemonic | Código em Hexadecimal |
|---|---|
| ADC | 0x00 |
| ADD | 0x01 |
| AND | 0x02 |
| BIC | 0x03 |
| CMN | 0x04 |
| CMP | 0x05 |
| EOR | 0x06 |
| MOV | 0x07 |
| MVN | 0x08 |
| ORR | 0x09 |
| RSB | 0x0A |
| RSC | 0x0B |
| SBC | 0x0C |
| SUB | 0x0D |
| TEQ | 0x0E |
| TST | 0x0F |

Tabela1.1: Codificação das instruções de processamento de dados.

| Condições | Descrição | Código Hexa |
|---|---|---|
| EQ | Equal | 0x00 |
| NE | Not Equal | 0x01 |
| CS | Unsigned higher, or same | 0x02 |
| CC | Unsigned lower | 0x03 |
| MI | Negative | 0x04 |
| PL | Positive or zero | 0x05 |
| VS | Overflow | 0x06 |
| VC | No Overflow | 0x07 |
| HI | Unsigned Higher | 0x08 |
| LS | Unsigned lower, or same | 0x09 |
| GE | Greater, or equal | 0x0A |
| LT | Less than | 0x0B |
| GT | Greater than | 0x0C |
| LE | Less than, or equal | 0x0D |
| AL | Always | 0x0E |

Tabela 1.2: Codificação do campo de instrução COND.

## 2.1 O que Faltou Implementar

Para esse trabalho, faltou implementar muitas outras instruções do ARM740T, bem como o comportamento das instruções de *branch* e das intruções de multiplicação. Também faltou fazer o tratamento do *carry* na função de add.

## 2.2 Dificuldades Encontradas

As dificuldades encontradas para o desenvolvimento desse trabalho foram em relação a documentação sobre o processador, principalmente no que diz respeito ao funcionamento do pipeline. Houve perda de tempo na procura de documento que abordasse o funcionamento do pipeline de modo compreensível. As informações do *datasheets* são um pouco complexas e não contém muitos detalhes sobre o processador. Também houve a dificuldade em relação ao tempo destinado a esse trabalho, pois foi necessário dedicar um tempo maior de estudo no projeto de graduação para resolver alguns problemas de implementação e escrita.

## 2.3 Conclusão

Apesar de ter implementado poucas instruções, foi observado que o ArchC realmente é uma ferramenta que facilita em muito o trabalho de simular processadores. Pois permite ao desenvolvedor se preocupar apenas com o comportamento do mesmo, ficando para a linguagem as questões de hardware.