VIII ERMAC-R3

5º Encontro Regional de Matemática Aplicada e Computacional
20 a 22-Novembro-2008
Universidade Federal do Rio Grande do Norte - Natal/RN

Modelagem de Microcontroladores em B

Valério Medeiros Jr. , **Stephenson Galvão**, **David Deharbe**
Departamento de Informática e Matemática Aplicada, DIMAP, UFRN,
59078-970, Natal, RN
E-mail: {valerio.jr, stepgalvao, deharbe}@gmail.com

Resumo

Este trabalho descreve resumidamente a modelagem das plataformas *8051*, *PIC* e *Z80*. Essas modelagens são construídas com o método B, o qual aplica conceitos lógicos e matemáticos para descrever as características das plataformas. Dessa forma, tais modelagens podem ser utilizadas no projeto de plataformas para documentação, construção de simuladores, verificação de sua consistência e ainda na verificação de *software* em nível de montagem, ou seja, *assembly*. A verificação em *assembly* é a utilidade mais interessante, pois permite o desenvolvimento de *software* fidedigno por construção, sendo de extrema importância para construção de *softwares* críticos embarcados. Enfim, o trabalho apresenta algumas das relevantes técnicas utilizadas na modelagem e descreve vários dos seus detalhes.

Palavras-chave

Métodos Formais, Método B, Microcontroladores.

1 Introdução

A construção de *softwares* seguros, está entre um dos cinco grandes desafios da Sociedade Brasileira de Computação. Procurando resolver esse problema, foi proposta em [3] uma abordagem [1] para o desenvolvimento de sistemas fidedignos por construção, na qual um sistema é especificado formalmente em vários níveis de abstração, sendo refinado formalmente até o *assembly* da plataforma alvo. Entretanto, para o completo desenvolvimento dessa abordagem, é necessário inicialmente ter-se, em B[1], a especificação da plataforma alvo. Assim, o objetivo desse trabalho é demonstrar como são desenvolvidas, em B, as modelagens das plataformas utilizando-se como estudo de caso as modelagens dos microcontroladores[2] *Intel 8051*, *PIC* e *Zilog Z80*, e dessa forma, enaltecer a abordagem citada.

As modelagens, por sua vez, foram desenvolvidas utilizando-se os mecanismos de divisão de responsabilidades providos pelo Método B. Assim, inicialmente foram desenvolvidos módulos básicos, nos quais os tipos de dados *bit*, vetor de *bits* e outros são especificados juntamente com suas operações. Depois de criada a biblioteca comum aos três microcontroladores, foram desenvolvidos módulos específicos a cada plataforma alvo, como o módulo responsável pela modelagem da memória, o responsável pelas operações da ALU (*Arithmetic Logic Unit*) e o que, utilizando-se dos outros módulos, define as instruções *assembly* da plataforma.

Com isso, o presente artigo está estruturado do seguinte modo. Inicialmente tem-se uma breve introdução ao Método B. Em seguida, é iniciada uma descrição geral da especificação dos microcontroladores direcionada aos seus conceitos comuns. As três seções seguintes contêm mais detalhes sobre a modelagem específica de cada microcontrolador. Ao final, é realizada a conclusão destacando quais os resultados e as vantagens da modelagem de microcontroladores utilizando-se o Método B.

2 Método B

O método B é uma metodologia formal para especificação de sistemas. Nele as definições especificadas podem ser estruturadas em módulos. Esses módulos são nomeados de acordo com seus níveis de abstração. Entretanto, este trabalho tem maior foco no nível inicial de abstração, conhecido como máquina abstrata.

Dessa forma, a especificação inicial de um sistema pode ser vista como a composição de várias máquinas abstratas, na qual cada uma é responsável por uma determinada parte do sistema, facilitando com isso, a compreensão, manutenção e o re-uso da especificação.

Para descrever os aspectos de um componente do sistema, as máquinas abstratas utilizam-se da Notação de Máquina Abstrata (NMA). A NMA é uma linguagem de especificação formal fundamentada em estados, em que cada especificação encapsula dados e operações que são restringidas pelo seu *invariant*. Assim, a base matemática de B é a lógica

[1] O leitor interessado em conhecer mais detalhes dessa abordagem pode visitar a página do projeto *B2Asm* em http://www.b2asm.googlecode.com

[2] Os microcontroladores são computadores simplificados e baratos, que em um único circuito integrado contém várias das unidades que compõem, por exemplo, um computador pessoal. Basicamente, microcontroladores são compostos por unidade de processamento, memória de acesso aleatório, memória de somente leitura e portas de entrada e saída.

```
MACHINE
  A8051
SEES
  TYPES, ALU,
INCLUDES
  MEMORY
CONCRETE_VARIABLES
  pc
INVARIANT
  pc ∈ INSTRUCTION
INITIALISATION
  pc := 0
OPERATIONS
AJMP(jump) =
  PRE
    jump ∈ INSTRUCTION
  THEN
    pc := jump
  END
...
```

Figura 1: Parte inicial da Máquina Abstrata do *8051*

de primeira ordem, aritmética de inteiros e a teoria dos conjuntos.

2.1 Estrutura da Máquina Abstrata

Observando o exemplo da máquina abstrata da figura 1 é possível identificar os seus principais elementos. O nome é definido na cláusula *MACHINE*. A declaração das variáveis de estados da especificação aparece na cláusula *VARIABLES*. O *INVARIANT* estabelece os estados válidos que a especificacão pode assumir, sendo o estado inicial definido na cláusula *INITIALISATION*. As operações, definidas em *OPERATIONS* são utilizadas para alterar ou consultar o estado da máquina, e a maioria das suas construções são semelhantes às das linguagens de programação tradicionais. Nessa figura temos uma simples operação ($AJMP$), que recebe o parâmetro *jump* e atribui seu valor na variável *pc*.

O método B também suporta várias formas de modularização da especificação, ou seja, uma máquina pode ser composta de conceitos de várias outras máquinas interdependentes. Basicamente, três cláusulas de modularização são utilizadas no método B. A cláusula $INCLUDES$ permite a máquina incluir instâncias de outras máquinas [1]. A cláusula $SEES$ referencia uma máquina, e permite acessar as suas constantes, funções e variáveis, porém sem modificá-las. A cláusula $USES$ permite o compartilhamento de leitura de dados entre máquinas incluídas, ou seja, referenciadas na cláusula $INCLUDES$.

Mais um exemplo da aplicação dessas cláusulas é ilustrado na figura 2. Nessa figura, a máquina $M1$ incluí instâncias de $M2$, $M3$ e $M4$. Além disso, a $M4$ acessa dados através de $SEES$ de uma máquina $M5$. O sentido das setas é o da máquina com cláusula de modularização para a máquina referenciada.

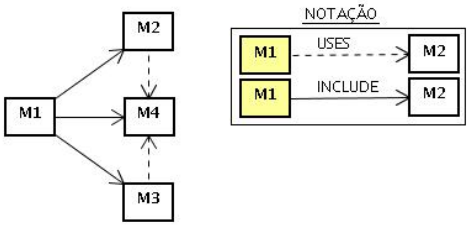

Figura 2: Notação das construções de modularização do Método B

3 Modelagem B dos Microcontroladores

Nesse trabalho, os microcontroladores selecionados para modelagem B foram *8051*, *PIC16C432* e *Z80*. Esses microcontroladores são bastante simples, têm um custo muito baixo, podem ser utilizados para executar *software* embarcado em diversos tipos de aplicações e têm uma boa difusão no mercado para sistemas embarcados, características que fundamentaram suas escolhas. Além disso, esses possuem uma complexidade inferior aos atuais processadores e, dessa forma, também servem como "modelo arcabouço" para futuras novas modelagens.

A construção dos modelos B desses microcontroladores é uma atividade complexa e custosa, e por isso várias técnicas são fundamentais para a construção de uma modelagem adequada. A técnica de modularização[3] é muito importante [2], principalmente com relação à modelagem das plataformas.

Uma modularização adequada proporciona uma série de benefícios. Alguns deles são a melhoria na capacidade de detecção de erros e verificação dos módulos, pois isto é realizado de forma mais independente, o que também simplifica o trabalho em equipe. Além disso, a mais evidente vantagem da modularização está na possibilidade de re-uso de módulos especificados anteriormente, diminuindo assim o esforço humano repetitivo.

Além de reduzir o esforço humano, o re-uso tem outro papel de destaque. Ele evita as redundâncias na modelagem e favorece a padronização do uso das funções. Dessa forma, uma modelagem que utiliza essa técnica, tem o seu processo de prova significativamente simplificado, uma vez que evita-se definições redundantes.

O uso intensivo dessas técnicas foi uma decisão de projeto recentemente adotada na modelagem B das plataformas citadas anteriormente. Assim, atualmente a modelagem mais modularizada é a *Intel 8051*, e portanto essa terá um maior foco neste trabalho.

[3] A modularização é uma das técnicas usadas em Engenharia de *Software* para o desenvolvimento de projetos de grande escala, em que o objetivo é separar e organizar responsabilidades em módulos de código com funções bem definidas e tão independentes quanto possível.

Atualmente as plataformas têm os seus módulos organizados nas seguintes categorias:

Módulo de tipos de dados é utilizado para definir os tipos de dados e as suas operações aritméticas e lógicas utilizadas nessas plataformas. Os módulos dessa categoria estão agrupados em uma biblioteca e organizados em duas subcategorias, módulos de dados primitivos e especializados. O primitivo define os tipos de dados mais básicos e comuns, *BIT_DEFINITION*, *BIT_VECTOR_DEFINITION* e *BYTE_DEFINITION*. O especializado define os tipos de dados mais complexos os quais se utilizam das definições dos tipos primitivos e são exclusivos de uma determinada plataforma, sendo especificados em *TYPES*.

Módulo de memória define o modelo dos dados de memória, constantes e operações para manipulação e acesso. Normalmente, os módulos dessa categoria é específico para cada plataforma e denominado *MEMORY*.

Módulo de operações lógicas e aritméticas define funções e constantes para manipulação dos dados de registradores e memórias. Essas funções são utilizadas nas definições das instruções da plataforma. Normalmente, os módulos dessa categoria é específico para cada plataforma e denominado *ALU*.

Módulo de instruções da plataforma define os elementos da plataforma e a ação de cada uma das instruções sobre o seu estado. Ele é o mais importante, pois é o módulo utilizado na metodologia proposta por [3] para verificar *software* em nível de montagem.

Os módulos dessas categorias são interrelacionados e, como exemplo, a figura 3 apresenta um diagrama das máquinas da plataforma *8051*. Mais detalhes sobre essas máquinas serão discutidos nas seções seguintes.

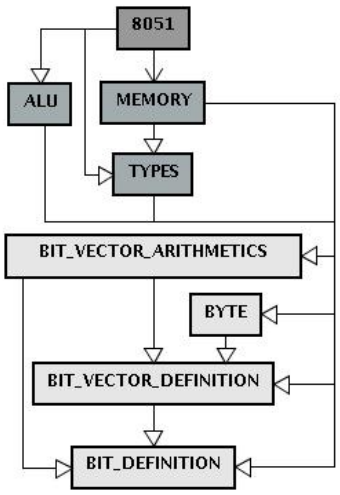

Figura 3: Estrutura de Módulos da plataforma *8051*

3.1 Definições gerais para conceitos de *Hardware*

A figura 3 apresenta uma forma conveniente para estruturar os conceitos de *hardware* em B. As máquinas em cinza claro dessa figura contêm definições comuns às várias plataformas. Essas definições são agrupadas na biblioteca *HARDWARE_LIBRARY*.

3.1.1 Definições para representar e manipular *bits*.

O módulo *BIT_DEFINITION* contém a definição do tipo *bit*, os seus operadores lógicos (negação, conjunção, disjunção, disjunção exclusiva) e a função de conversão de *booleano* para *bits*. Primeiramente, *bits* são definidos como um conjunto de inteiros ({0,1}):

$BIT = 0..1$

A operação de negação é uma função unária definida como:

$bit\_not \in BIT \rightarrow BIT \wedge$
$\forall(bb).(bb \in BIT \implies bit\_not(bb) = 1 - bb)$

O módulo também provê lemas sobre a negação que podem ser úteis aos usuários da biblioteca para desenvolver provas.

$bit\_not(0) = 1;$
$bit\_not(1) = 0;$
$\forall(bb).(bb \in BIT \implies bit\_not(bit\_not(bb)) = bb);$

A conjunção é função binária definida como:

$bit\_and \in BIT \times BIT \rightarrow BIT \wedge$
$\forall(b1, b2).(b1 \in BIT \wedge b2 \in BIT \implies$
$((bit\_and(b1, b2) = 1) \iff (b1 = 1) \wedge (b2 = 1)))$

O módulo também contém a definição de *bit_or* (disjunção) e *bit_xor*(disjunção exclusiva), como também os lemas para essas operações. Enfim, a conversão de *booleanos* para *bits* é simplesmente definida como:

$bool\_to\_bit \in \mathbf{BOOL} \rightarrow BIT \wedge$
$bool\_to\_bit = \{\mathbf{TRUE} \mapsto 1, \mathbf{FALSE} \mapsto 0\}$

Todos os lemas providos neste módulo foram provados automaticamente através do provador de teoremas incluído no ambiente de desenvolvimento B. Assim, nenhuma das provas necessitou de intervenção humana.

3.1.2 Representação e manipulação de vetor de *bits*

Seqüências são pré-definidas em B, como uma função em que o domínio é um intervalo de inteiros maiores que zero. Vetores de *bits* foram definidos como uma seqüência não vazia de *bits*, e *BIT_VECTOR* é o conjunto de todas as seqüências semelhantes.

$BIT\_VECTOR = \mathbf{seq1}(BIT)$

A função *bv_size* retorna o tamanho de um dado vetor de *bits*. Isto é basicamente um encapsulamento para uma função pré-definida *size*.

$bv\_size \in BIT\_VECTOR \rightarrow \mathbb{N}_1 \wedge$
$bv\_size = \lambda bv \bullet (bv \in BIT\_VECTOR \mid \mathbf{size}(bv))$

Outras duas funções definidas semelhantemente foram *bv_set* e *bv_clear*, que dado um vetor de *bits* e uma posição do vetor de *bits*, retorna o vetor de

bits com o *bit* correspondente modificado para 1 ou 0. Somente a primeira definição é mostrada aqui.

$bv\_set \in BIT\_VECTOR \times \mathbb{N} \rightarrow BIT\_VECTOR \wedge$
$bv\_set = \lambda v, n \bullet (v \in BIT\_VECTOR \wedge$
$n \in \mathbb{N} \wedge n < bv\_size(v) \mid v \oplus \{n + 1 \mapsto 1\})$

A função *bv_catenate* recebe como parâmetros dois vetores de *bit*, v e w, e retorna a concatenação de v e w, sendo que v constitui a parte mais significativa do resultado.

$bv\_catenate \in$
$(BIT\_VECTOR \times BIT\_VECTOR \rightarrow BIT\_VECTOR)$
$\wedge bv\_catenate =$
$\lambda v, w \bullet (v \in BIT\_VECTOR \wedge$
$w \in BIT\_VECTOR \mid vw)$

Uma função *bv_zero* foi definida para que, dado um inteiro positivo n, retorne um vetor de *bits* do tamanho de n, com todos os *bits* valorados em 0. Uma função similar, chamada *bv_one*, com todos os *bits* valorados em 1, é também definida, mas não apresentada aqui.

$bv\_zero \in \mathbb{N}_1 \rightarrow BIT\_VECTOR \wedge$
$bv\_zero = \lambda n \bullet (n \in \mathbb{N}_1 \mid 1..n \times \{0\})$

Adicionalmente, o módulo provê definições de funções lógicas para combinar vetores de *bits*: *bit_not*, *bv_and*, *bv_or* e *bv_xor*. Somente as duas primeiras funções são apresentadas. Observe que o domínio dos operadores binários é restrito aos pares de vetores de *bits* de mesmo tamanho.

$bv\_not \in BIT\_VECTOR \rightarrow BIT\_VECTOR \wedge$
$bv\_not = \lambda v \bullet (v \in BIT\_VECTOR \mid$
$\lambda i \bullet 0..bv\_size(v) - 1 \mid bit\_not(v(i))) \wedge$
$bv\_and \in$
$BIT\_VECTOR \times BIT\_VECTOR \rightarrow BIT\_VECTOR$
$\wedge bv\_and = \lambda v_1, v_2 \bullet (v_1 \in BIT\_VECTOR \wedge$
$v_2 \in BIT\_VECTOR \wedge$
$bv\_size(v_1) = bv\_size(v_2) \mid \lambda i \bullet 0..bv\_size(v_1) - 1 \mid$
$bit\_and(v_1(i), v_2(i)))$

Vários lemas foram providos nas operações de vetores de *bits*, mas não são aqui apresentados. Estes lemas expressam propriedades referentes ao tamanho do resultado das funções, como também as propriedades da álgebra clássica, tais como, a associatividade e a comutatividade.

3.1.3 Modelando *bytes*

Vetores de *bit* de tamanho 8 são *bytes*. Eles formam uma entidade no projeto de *hardware*. Basicamente, as seguintes definições as seguintes definições são necessárias:

$BYTE\_WIDTH = 8 \wedge$
$BYTE\_INDEX = 0..(BYTE\_WIDTH - 1) \wedge$
$BYTE \subseteq BIT\_VECTOR \wedge$
$BYTE = \{v \in BIT\_VECTOR \wedge$
$bv\_size(v) = BYTE\_WIDTH\} \wedge$
$BYTE\_ZERO \in BYTE \wedge$
$BYTE = BYTE\_INDEX \times \{0\}$

3.1.4 Aritmética de Vetor de *Bits*.

Vetores de *bit* costuman representar e combinar números de intervalos de inteiros e ponto flutuante. Esse módulo define uma função *bv_to_nat* que mapeia vetores de *bits* para números naturais:

$bv\_to\_nat \in BIT\_VECTOR \rightarrow \mathbb{N} \wedge$
$bv\_to\_nat = \lambda v \bullet (v \in BIT\_VECTOR$
$\mid \sum i \bullet (i \in \mathrm{dom}(v) \bullet v(i) \times 2^i))$

4 *Intel 8051*

O microcontrolador *8051* é o membro originário de sua família servindo assim como base para os demais dispositivos que a compõem, como o *8751* e o *8052*. Suas características básicas são: possui uma *CPU* de 8 *bits*, faz a separação entre memória de dados e memória de programas, e possui um conjunto de instruções completo. A memória de dados é organizada em dois blocos, um para os registradores comuns, e outro para os de funções especiais, possuindo 128 *bytes* cada um; já as suas instruções estão classificadas como aritméticas, lógicas, de transferência de dados e de desvio de controle, formando ao todo 256 instruções.

O módulo de dados especiais do *8051* possui basicamente os seguintes tipos: *INSTRUCTION*, responsável pela faixa de valores permitidos ao contador de programa[4]; o tipo *UCHAR*, conjunto dos números não negativos que podem ser representados por um *byte*; e o tipo *SCHAR*, números inteiros em complemento de dois representados por um *byte*.

Na máquina *MEMORY* é especificada a memória de dados do *8051*. Assim, para dividir essa memória em dois blocos de registradores, foram criados inicialmente dois conjuntos de endereços, o primeiro, denominado *RAM_ADDR*, contém os endereço de memória dos registradores de funções gerais; e o segundo, denominado *SFR_ADDR*, é formado pelos endereços de memória dos registradores de funções especiais. A união desses dois conjuntos dá origem ao conjunto denominado *MEM_ADDR*, que representa todos os endereços de memória de dados interna possíveis no *8051*, sendo o total de 256 endereços divididos em blocos de 128, um para cada conjunto de registradores.

Depois de criados os endereços de memória, é necessário agora relacionar tais endereços com as estruturas de representação de dados. No *8051*, essa ligação é realizada através da variável de estado *mem*, a qual representa uma função total entre os endereços de memória e um conjunto *byte*, sendo especificada da seguinte maneira: $mem \in MEM\_ADDR \rightarrow BYTE$. Uma vez criado o estado da máquina, é necessário especificar as operações que o altere. Assim, no 8051 as operações servem como um mecanismo que altera a memória do controlador atualizando os valores nos endereços de memória.

A especificação da unidade lógica e aritmética, ou em inglês *Arithmetic Logic Unit* (ALU), da plataforma é realizada na máquina ALU. Nela, são criadas funções de acordo com as operações aritméticas e lógicas necessária ao *8051*. Assim na ALU podemos encontrar operações lógicas, operação aritméticas e operações de cópia e transferência de dados. Como exemplo, temos, abaixo, a função *swap* que recebe um *byte* e faz a inversão dos 4 *bit* mais significativos com os 4 *bits* menos

[4] O contador de programas é um registrador utilizado pela plataforma para controlar o fluxo de execução das instruções.

significativo.

$$
\begin{array}{l}
swap \in BYTE \longrightarrow BYTE \\
\wedge swap = \lambda(bt).(bt \in BYTE \mid \{0 \mapsto bt(4), 1 \mapsto bt(5), \\
\qquad\qquad 2 \mapsto bt(6), 3 \mapsto bt(7), 4 \mapsto bt(0), \\
\qquad\qquad 5 \mapsto bt(1), 6 \mapsto bt(2), 7 \mapsto bt(3)\})
\end{array}
$$

Por fim, utilizando as máquinas anteriormente citadas, o componente *A8051* especifica o conjunto de instruções presente no microcontrolador. Um exemplo da especificação de uma instrução pode ser visto em seguida, no qual a instrução *swap* é especificada utilizando-se da função *swap*, definida na *ALU*, e da operação *addrSetDirect*, especificada em *MEMORY* e responsável por atualizar o valor (*swap(mem(addr))*) de um endereço de memória.

$$
\begin{array}{l}
SWAP(addr) = \\
\quad \textbf{PRE} \\
\quad\quad addr : MEM\_ADDR \\
\quad \textbf{THEN} \\
\quad\quad addrSetDirect(addr, swap(mem(addr))) \\
\quad \textbf{END}
\end{array}
$$

5 *Microchip PIC*

Os microcontroladores da família *PIC* possuem um certo número de características em comum, como um reduzido conjunto de instruções e a divisão entre memória de dados e memória de programas. Entretanto, alguns aspectos, como o tamanho da memória , variam de acordo com o modelo. Devido a isso, o modelo aqui apresentando tem como objetivo ser o mais genérico possível. Contudo, algumas características específicas necessárias foram implementadas de acordo com o modelo *PIC16C432* que possui um processador de 8 *bits* e endereços com 9 *bits* de largura.

A especificação do *PIC*, assim como no *8051*, foi distribuída entre quatro máquinas: *TYPES*, *MEMORY*, *ALU* e *PIC*. O módulo de tipos especiais *MEMORY* possui os mesmos tipos que o *8051*, exceto pelo tipo *SCHAR*, que não foi necessário na especificação do *PIC*. Também, assim como no *8051*, nesse modulo foram especificadas funções responsáveis pela a transformação entres os tipos utilizados pelo *PIC*.

A memória de dados do *PIC* é formada por 4 bancos de 128 palavras cada. Os endereços das palavras possuem o tamanho de 9 *bits*, sendo os dois *bits* mais significativos o endereço do banco, e os outros sete *bits* a localização do endereço no banco. Desse modo, a memória do *PIC* foi especificada como demonstra o código abaixo. O tipo *BANK*, referente aos bancos de endereços, é especificado como sendo um vetor de dois *bits* e o tipo *LOCATION*, referente ao endereço da memória dentro do banco, é implementada como sendo um vetor de sete *bits*. No final, temos o tipo *ADDR*, referente ao endereço de memória, como sendo a união do quantidade de bancos com a quantidade de endereços em cada banco.

$$
\begin{array}{l}
BANK\_WIDTH = 2 \\
\wedge BANK \subset BIT\_VECTOR \\
\wedge BANK = \{vv \mid vv \in BIT\_VECTOR \\
\qquad \wedge bv\_size(vv) = BANK\_WIDTH\} \\
\wedge LOCATION\_WIDTH = 7 \\
\wedge LOCATION \subset BIT\_VECTOR \\
\wedge LOCATION = \{vv \mid vv \in BIT\_VECTOR \\
\qquad \wedge bv\_size(vv) = LOCATION\_WIDTH\} \\
\wedge BANK\_SIZE = 2^{LOCATION\_WIDTH}
\end{array}
$$

A *ALU* do *PIC* é feita de forma similar à dos outros microcontroladores, possuindo funções referentes as operações lógicas e aritméticas necessárias na implementação do *PIC*. Como exemplo de uma função especificada na *ALU* temos a função *bitget*, demonstrada a seguir. Nela são passados um *byte* e uma posição de um *bit* como parâmetro, após isso é definido que o retorno da função é o *bit* localizado na posição informada.

$$
\begin{array}{l}
bitget \in (BYTE * BYTE\_INDEX)BIT \\
\wedge \lambda(ww, ii).(ww \in BYTE \wedge ii \in BYTE\_INDEX \\
\qquad\qquad \mid bitget(ww, ii) = ww(ii))
\end{array}
$$

Por fim, temos o módulo onde são implementadas as instruções do microcontrolador, no qual está localizado a instrução *BTFSS*, especificada como no código abaixo. Ela recebe uma posição de memória e a posição de um *bit* como argumentos, e caso o *bit* da posição informada seja 1, o contador de programa (*pc*) é incrementado em duas posições, caso seja 0, o *pc* é incrementado em uma posição.

$$
\begin{array}{l}
BTFSS(ff, bb) = \\
\quad \textbf{PRE } ff : REGISTER \wedge bb : BYTE\_INDEX \\
\quad \textbf{THEN} \\
\quad\quad \textbf{IF } bitget(mem(ff), bb) = 1 \textbf{ THEN} \\
\quad\quad\quad pc := instruction\_next(instruction\_next(pc)) \\
\quad\quad \textbf{ELSE} \\
\quad\quad\quad pc := instruction\_next(pc) \\
\quad\quad \textbf{END} \\
\quad \textbf{END}
\end{array}
$$

6 *Zilog Z80*

O *Z80* foi um microcontrolador importante desenvolvido em 1976 pela Zilog. Nessa época ele obteve um grande destaque, pois suportava todo conjunto de instruções do *8080*, um dos antecessores dos atuais *Pentium* da *Intel Corporation*, e oferecia muitas melhorias comparado ao *8080*. Algumas das suas melhorias foram: o aprimoramento no conjunto de instruções, a criação de novos registradores auxiliáres, e um menor consumo energético. Além disso, ele tinha custo inferior e outras características interessantes.

O *Z80* suporta 158 diferentes instruções incluindo todas as 78 do 8080. Ele contém 208 *bits* de Leitura/Escrita em memória, que são disponíveis para o programador. Os seus principais registradores são 2 grupos de 6 registradores de propósitos gerais, que podem ser usados individualmente com 8 *bits* ou em

par com 16 *bits*. Existem também dois registradores de acumulador e de *flag*, e seis registradores de propósitos especiais. Ele também tem registradores adicionais para controle da pilha, contagem de programa, indexação de dados, controle de atualização de dados e gerenciamento de interrupções. Adicionalmente, é suportado por uma extensa família de periféricos controladores.

A seguir são apresentadas partes das definições do *Z80*. Como os demais modelos, a máquina *Z80* é modularizada, por isso ela incluí *MEMORY* e tem uma referência (*SEES*) para as definições em *TYPES* e *ALU*. Ela também defini dois conjuntos (*id_reg_8* e *id_reg_16*) que representam os nomes dos registradores da plataforma.

$$id\_reg\_8 = \{a0, f0, f_0, a_0, b0, c0, b_0, c_0, d0, e0, d_0, e_0, h0, l0, h_0, l_0, i0, r0\};$$
$$id\_reg\_16 = \{BC, DE, HL, SP, AF\}$$

Além disso, existem as variáveis que representam o estado das portas de entrada e saída, pilha, registradores para endereçamento indireto, para propósitos gerais de 8 e 16 *bits*, controle de atualização de dados e interrupções. Na cláusula *INVARIANT*, é definido matematicamente o tipo de cada um dessas variáveis.

$$stack \in BV16 \rightarrow UCHAR \wedge$$
$$rgs8 \in id\_reg\_8 \rightarrow UCHAR$$
$$pc \in BV16 \wedge sp \in BV16 \wedge$$
$$ix \in BV16 \wedge iy \in BV16 \wedge$$
$$stack \subset mem \wedge$$
$$i\_o\_ports \in (CHAR \rightarrow UCHAR)$$

Enfim, a abordagem de especificação utilizada na modelagem automatizou bastante o seu processo de prova, pois a grande maioria das obrigações de provas foram realizadas automaticamente.

7 Conclusão

A necessidade comercial para produzir *hardware* e *software* de alta qualidade é sempre crescente. Métodos formais já têm demonstrado sucesso na especificação de *softwares* comerciais e críticos; protocolos padrão e projetos de *hardware*.

O suporte fornecido pelo método B é bastante adequado, maduro, extensivo e eficaz no processo de especificação formal dessas plataformas. Isso vem sendo perceptível, pois até então, todas as características relevantes das plataformas foram modeladas de forma apropriada, principalmente as importantes para verificação em nível de montagem (*assembly*). Por outro lado, uma das qualidades de destaque do método B é relacionada com a boa capacidade de verificação automática das ferramentas, pois, na versão estável das modelagens, do total de 503 provas dos módulos principais do *PIC*, *8051* e *Z80* apenas 3 referentes ao *Z80* foram provadas de forma semi-automática. Ou seja, desse total de provas menos de 1% necessitaram de uma simples intervenção do projetista para ser concluída.

Adicionalmente, a aplicação das técnicas modularização e re-uso tornou o processo de especificação das plataformas mais padronizado e acelerado. Desse modo, é demonstrada a viabilidade dessas especificações, pois com a aquisição de maturidade das ferramentas e do projetista, o processo de modelagem ganhou agilidade e maior automação na fase de validação do projeto.

Enfim, a modelagem B dessas plataformas possibilitarão a verificação em um dos mais altos níveis de certificação de *software*. Não somente, elas podem ser importantes no projeto de plataformas para a construção dos simuladores, documentação e sua própria verificação formal, permitindo identificar várias inconsistências delas em tempo de projeto.

Referências

[1] J. R. Abrial. The B Book: Assigning Programs to Meanings. 1. ed. United States of America: Cambridge University Press, 1996.

[2] D. Bell, Ian Morrey e John Pugh, "Software Engineering: A Programming Approach", segunda edição, Prentice Hall, Nova Iorque, 1992.

[3] B. Dantas, D. Deharbe, S.Galvão, et al, Proposta e Avaliação de uma Abordagem de Desenvolvimento de Software Fidedigno por Construção com o Método B, SEMISH, 2008, Belém.

[4] Microchip. PIC16C432 - Data Sheet OTP 8-Bit CMOS MCU with LIN Transceiver. USA, 2002. 197p.

[5] D.E.C. Nicolosi, "Microcontrolador 8051 detalhado",Ed. Érica, São Paulo, 2005.

[6] B. William, "Z80 microcomputer design projects",Ed. Sams, 1980.