Разработка модели микропроцессорного ядра в системе визуально-имитационного моделирования MATLAB/Simulink с блоком обработки прерываний

Андрей СТРОГОНОВ, д. т. н.
andreis@hotmail.ru
Артем БУСЛОВ
buslov-artem@yandex.ru
Максим МОТЫЛЕВ
polikort@rambler.ru

Данная статья продолжает цикл работ по изучению возможностей системы визуально-имитационного моделирования MATLAB/Simulink для проектирования микропроцессорных ядер с последующей реализацией в базисе ПЛИС [1–3]. Здесь предлагается модифицировать систему команд микропроцессорного ядра из работы [4], с целью разработки новых блоков, проектирование которых не рассматривалось в [1–3]: обработка прерываний, регистр флагов, ОЗУ, порт ввода/вывода. Модель процессора имеет RISC-архитектуру (процессор с сокращенным набором команд, инструкции одинаковой длины).

Команды, необходимые для выполнения базовых манипуляций с данными (перенос данных между ПЗУ, регистрами, ОЗУ и стеком, арифметические и логические операции, операции сравнения, операции условных и безусловного переходов, работа с подпрограммами, операции обработки прерываний), представлены в таблице 1.

С помощью графического редактора моделей Simulink построена модель 8-разрядного микропроцессорного ядра с гарвардской архитектурой (рисунок). Процессор оперирует с числами

Таблица 1. Система команд микропроцессорного ядра

Машинный код команды в шестнадцатиричной системе счисления	Код команды в десятичной системе счисления	Мнемоника команды	Описание команды
0000H	0	NOP	Нет операции
0001H	1	RET	Возврат из подпрограммы
0002H	2	INTON	Разрешить прерывания
0003H	3	INTOFF	Запретить прерывания
0004H	4	PUSHF	Сохранение регистра флагов в стеке
0005H	5	POPF	Восстановление регистра флагов из стека
0006H	6	PUSH A	Сохранение значения регистра A в стеке
0007H	7	PUSH B	Сохранение значения регистра B в стеке
0008H	8	PUSH C	Сохранение значения регистра C в стеке
0009H	9	PUSH D	Сохранение значения регистра D в стеке
000aH	10	PUSH E	Сохранение значения регистра E в стеке
000bH	11	POP A	Восстановление значения регистра A из стека
000cH	12	POP B	Восстановление значения регистра B из стека
000dH	13	POP C	Восстановление значения регистра C из стека
000eH	14	POP D	Восстановление значения регистра D из стека
000fH	15	POP E	Восстановление значения регистра E из стека
0010H	16	IN A	Прочитать данные из порта и записать в регистр A
0011H	17	IN B	Прочитать данные из порта и записать в регистр B
0012H	18	IN C	Прочитать данные из порта и записать в регистр C
0013H	19	IN D	Прочитать данные из порта и записать в регистр D
0014H	20	IN E	Прочитать данные из порта и записать в регистр E
0015H	21	OUT A	Вывести значение регистра A в порт
0016H	22	OUT B	Вывести значение регистра B в порт
0017H	23	OUT C	Вывести значение регистра C в порт
0018H	24	OUT D	Вывести значение регистра D в порт
0019H	25	OUT E	Вывести значение регистра E в порт
001aH	26	DEC A	Декремент регистра A (вычитание 1)
001bH	27	DEC B	Декремент регистра B (вычитание 1)
001cH	28	DEC C	Декремент регистра C (вычитание 1)
001dH	29	DEC D	Декремент регистра D (вычитание 1)
001eH	30	DEC E	Декремент регистра E (вычитание 1)
001fH	31	INC A	Инкремент регистра A (прибавление 1)
0020H	32	INC B	Инкремент регистра B (прибавление 1)
0021H	33	INC C	Инкремент регистра C (прибавление 1)
0022H	34	INC D	Инкремент регистра D (прибавление 1)
0023H	35	INC E	Инкремент регистра E (прибавление 1)
0024H	36	ADD A,B	Сложение значений в регистрах A и B, результат помещается в A
0025H	37	ADD A,C	Сложение значений в регистрах A и C, результат помещается в A
0026H	38	ADD A,D	Сложение значений в регистрах A и D, результат помещается в A
0027H	39	ADD A,E	Сложение значений в регистрах A и E, результат помещается в A
0028H	40	SUB A,B	Вычитание значений в регистрах A и B, результат помещается в A
0029H	41	SUB A,C	Вычитание значений в регистрах A и C, результат помещается в A
002aH	42	SUB A,D	Вычитание значений в регистрах A и D, результат помещается в A
002bH	43	SUB A,E	Вычитание значений в регистрах A и E, результат помещается в A
002cH	44	AND A,B	Побитное логическое И значений в регистрах A и B, результат помещается в A
002dH	45	AND A,C	Побитное логическое И значений в регистрах A и C, результат помещается в A
002eH	46	AND A,D	Побитное логическое И значений в регистрах A и D, результат помещается в A
002fH	47	AND A,E	Побитное логическое И значений в регистрах A и E, результат помещается в A
0030H	48	OR A,B	Побитное логическое ИЛИ значений в регистрах A и B, результат помещается в A
0031H	49	OR A,C	Побитное логическое ИЛИ значений в регистрах A и C, результат помещается в A
0032H	50	OR A,D	Побитное логическое ИЛИ значений в регистрах A и D, результат помещается в A

Таблица 1. Система команд микропроцессорного ядра (окончание)

Машинный код команды в шестнадцатиричной системе счисления	Код команды в десятичной системе счисления	Мнемоника команды	Описание команды
0033H	51	OR A,E	Побитное логическое ИЛИ значений в регистрах A и E, результат помещается в A
0034H	52	XOR A,B	Побитное логическое ИСКЛЮЧАЮЩЕЕ ИЛИ значений в регистрах A и B, результат помещается в A
0035H	53	XOR A,C	Побитное логическое ИСКЛЮЧАЮЩЕЕ ИЛИ значений в регистрах A и C, результат помещается в A
0036H	54	XOR A,D	Побитное логическое ИСКЛЮЧАЮЩЕЕ ИЛИ значений в регистрах A и D, результат помещается в A
0037H	55	XOR A,E	Побитное логическое ИСКЛЮЧАЮЩЕЕ ИЛИ значений в регистрах A и E, результат помещается в A
0038H	56	MOV A,B	Загрузка в регистр A значения, содержащегося в регистре B
0039H	57	MOV A,C	Загрузка в регистр A значения, содержащегося в регистре C
003aH	58	MOV A,D	Загрузка в регистр A значения, содержащегося в регистре D
003bH	59	MOV A,E	Загрузка в регистр A значения, содержащегося в регистре E
003cH	60	MOV B,A	Загрузка в регистр B значения, содержащегося в регистре A
003dH	61	MOV B,C	Загрузка в регистр B значения, содержащегося в регистре C
003eH	62	MOV B,D	Загрузка в регистр B значения, содержащегося в регистре D
003fH	63	MOV B,E	Загрузка в регистр B значения, содержащегося в регистре E
0040H	64	MOV C,A	Загрузка в регистр C значения, содержащегося в регистре A
0041H	65	MOV C,B	Загрузка в регистр C значения, содержащегося в регистре B
0042H	66	MOV C,D	Загрузка в регистр C значения, содержащегося в регистре D
0043H	67	MOV C,E	Загрузка в регистр C значения, содержащегося в регистре E
0044H	68	MOV D,A	Загрузка в регистр D значения, содержащегося в регистре A
0045H	69	MOV D,B	Загрузка в регистр D значения, содержащегося в регистре B
0046H	70	MOV D,C	Загрузка в регистр D значения, содержащегося в регистре C
0047H	71	MOV D,E	Загрузка в регистр D значения, содержащегося в регистре E
0048H	72	MOV E,A	Загрузка в регистр E значения, содержащегося в регистре A
0049H	73	MOV E,B	Загрузка в регистр E значения, содержащегося в регистре B
004aH	74	MOV E,C	Загрузка в регистр E значения, содержащегося в регистре C
004bH	75	MOV E,D	Загрузка в регистр E значения, содержащегося в регистре D
004cH	76	CMP A,B	Команда сравнения: из A вычитается B, выставляются флаги, но результат никуда не записывается
004dH	77	CMP A,C	Команда сравнения: из A вычитается C, выставляются флаги, но результат никуда не записывается
004eH	78	CMP A,D	Команда сравнения: из A вычитается D, выставляются флаги, но результат никуда не записывается
004fH	79	CMP A,E	Команда сравнения: из A вычитается E, выставляются флаги, но результат никуда не записывается
0050H	80	CMP B,A	Команда сравнения: из B вычитается A, выставляются флаги, но результат никуда не записывается
0051H	81	CMP B,C	Команда сравнения: из B вычитается C, выставляются флаги, но результат никуда не записывается
0052H	82	CMP B,D	Команда сравнения: из B вычитается D, выставляются флаги, но результат никуда не записывается
0053H	83	CMP B,E	Команда сравнения: из B вычитается E, выставляются флаги, но результат никуда не записывается
0054H	84	CMP C,A	Команда сравнения: из C вычитается A, выставляются флаги, но результат никуда не записывается
0055H	85	CMP C,B	Команда сравнения: из C вычитается B, выставляются флаги, но результат никуда не записывается
0056H	86	CMP C,D	Команда сравнения: из C вычитается D, выставляются флаги, но результат никуда не записывается
0057H	87	CMP C,E	Команда сравнения: из C вычитается E, выставляются флаги, но результат никуда не записывается
0058H	88	CMP D,A	Команда сравнения: из D вычитается A, выставляются флаги, но результат никуда не записывается
0059H	89	CMP D,B	Команда сравнения: из D вычитается B, выставляются флаги, но результат никуда не записывается
005aH	90	CMP D,C	Команда сравнения: из D вычитается C, выставляются флаги, но результат никуда не записывается
005bH	91	CMP D,E	Команда сравнения: из D вычитается E, выставляются флаги, но результат никуда не записывается
005cH	92	CMP E,A	Команда сравнения: из E вычитается A, выставляются флаги, но результат никуда не записывается
005dH	93	CMP E,B	Команда сравнения: из E вычитается B, выставляются флаги, но результат никуда не записывается
005eH	94	CMP E,C	Команда сравнения: из E вычитается C, выставляются флаги, но результат никуда не записывается
005fH	95	CMP E,D	Команда сравнения: из E вычитается D, выставляются флаги, но результат никуда не записывается
0060H	96	XCHG A,B	Обмен местами значений в регистрах A и B
0061H	97	XCHG A,C	Обмен местами значений в регистрах A и C
0062H	98	XCHG A,D	Обмен местами значений в регистрах A и D
0063H	99	XCHG A,E	Обмен местами значений в регистрах A и E
0064H	100	XCHG B,C	Обмен местами значений в регистрах B и C
0065H	101	XCHG B,D	Обмен местами значений в регистрах B и D
0066H	102	XCHG B,E	Обмен местами значений в регистрах B и E
0067H	103	XCHG C,D	Обмен местами значений в регистрах C и D
0068H	104	XCHG C,E	Обмен местами значений в регистрах C и E
0069H	105	XCHG D,E	Обмен местами значений в регистрах D и E
01xxH	256–511	JMP	Безусловный переход по адресу xx
02xxH	512–767	JZ xx	Переход к адресу xx, если флаг ZF = 1
03xxH	768–1023	JNZ xx	Переход к адресу xx, если флаг ZF = 0
04xxH	1024–1279	JO xx	Переход к адресу xx, если флаг OF = 1
05xxH	1280–1535	JNO xx	Переход к адресу xx, если флаг OF = 0
06xxH	2816–3071	CALL	Вызов подпрограммы по адресу, заданному младшим байтом команды
07xxH	1536–1791	CMP A,xx	Сравнение содержимого регистра A и числа xx. При этом вычисляются флаги
08xxH	1792–2047	CMP B,xx	Сравнение содержимого регистра B и числа xx. При этом вычисляются флаги
09xxH	2048–2303	CMP C,xx	Сравнение содержимого регистра C и числа xx. При этом вычисляются флаги
0axxH	2304–2559	CMP D,xx	Сравнение содержимого регистра D и числа xx. При этом вычисляются флаги
0bxxH	2560–2815	CMP E,xx	Сравнение содержимого регистра E и числа xx. При этом вычисляются флаги
0cxxH	3072–3327	MOV A,xx	Непосредственная загрузка в регистр A значения, заданного младшим байтом команды
0dxxH	3328–3583	MOV B,xx	Непосредственная загрузка в регистр B значения, заданного младшим байтом команды
0exxH	3584–3839	MOV C,xx	Непосредственная загрузка в регистр C значения, заданного младшим байтом команды
0fxxH	3840–4095	MOV D,xx	Непосредственная загрузка в регистр D значения, заданного младшим байтом команды
10xxH	4096–4351	MOV E,xx	Непосредственная загрузка в регистр E значения, заданного младшим байтом команды
11xxH	4352–4607	MOV A,[xx]	Непосредственная загрузка в регистр A содержимого ОЗУ по адресу xx
12xxH	4608–4863	MOV B,[xx]	Непосредственная загрузка в регистр B содержимого ОЗУ по адресу xx
13xxH	4864–5119	MOV C,[xx]	Непосредственная загрузка в регистр C содержимого ОЗУ по адресу xx
14xxH	5120–5375	MOV D,[xx]	Непосредственная загрузка в регистр D содержимого ОЗУ по адресу xx
15xxH	5376–5631	MOV E,[xx]	Непосредственная загрузка в регистр E содержимого ОЗУ по адресу xx
16xxH	5632–5887	MOV [xx],A	Непосредственная загрузка в ячейку ОЗУ по адресу xx значения регистра A
17xxH	5888–6143	MOV [xx],B	Непосредственная загрузка в ячейку ОЗУ по адресу xx значения регистра B
18xxH	6144–6399	MOV [xx],C	Непосредственная загрузка в ячейку ОЗУ по адресу xx значения регистра C
19xxH	6400–6655	MOV [xx],D	Непосредственная загрузка в ячейку ОЗУ по адресу xx значения регистра D
1axxH	6656–6911	MOV [xx],E	Непосредственная загрузка в ячейку ОЗУ по адресу xx значения регистра E

в формате uint8 (целые положительные десятичные числа, представленные с 8-разрядной точностью). Ядро содержит следующие блоки: арифметико-логическое устройство (АЛУ, ALU); постоянное запоминающее устройство (ПЗУ, ROM); оперативное запоминающее устройство (ОЗУ, OZU); стек (Stack); 8 регистров общего назначения (POH, RON); 3 регистра специального назначения (PCH, RSN); порт ввода/вывода (Port); блок обработки прерываний (Block Int); разделитель полей команды (CmdCut).

Блок Signal Builder (рисунок) — виртуальная кнопка. В заданное время блок генерирует импульс прямоугольной формы. Выходное значение этого блока зависит от времени и определяется в свойствах компонента Signal Builder. Сигнал с блока Signal Builder поступает в блок обработки прерываний. На схеме (рисунок) присутствуют элементы задержки (Unit Delay), которые необходимы для того, чтобы модель при симулировании не зацикливалась (ошибка Arithmetic Loop), то есть чтобы в каждый момент времени производился только один цикл расчета выходных значений со всех блоков.

В ПЗУ хранится программа, которую выполняет процессор. В ПЗУ в качестве входного параметра поступает текущий адрес программы, выходным параметром является инструкция в числовом формате uint16 (целое положительное десятичное число, представленное с 16-разрядной точностью).

ПЗУ построено с использованием компонента Lookup Table (Lookup ROM), который представляется одномерным массивом или таблицей с двумя колонками, согласно которым всем возможным входным значениям соответствуют некоторые выходные значения, определенные самой программой (рис. 2 в [1]).

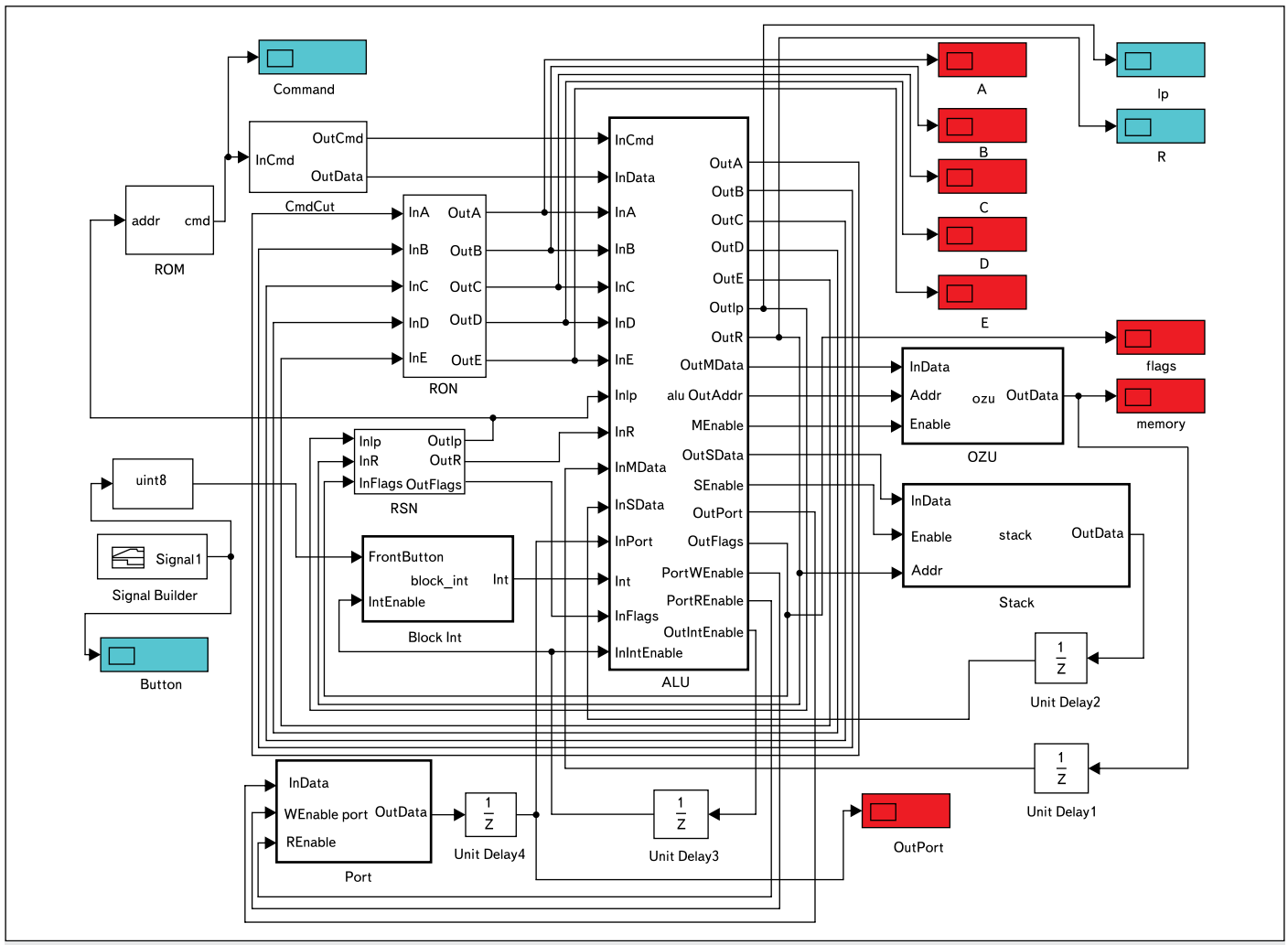

Рисунок. Модель 8-разрядного микропроцессорного ядра в системе MATLAB/Simulink

Программа записывается в ПЗУ в виде последовательности десятичных чисел, полученных посредством перевода машинных инструкций из шестнадцатиричной системы счисления в десятичную при помощи системы команд (табл. 1).

Процессор может манипулировать только 8-разрядными данными, а в ПЗУ инструкции хранятся в виде 16-разрядных слов, поэтому необходимо разделить двухбайтное число на два однобайтных: старшее и младшее. Это происходит в блоке разделения полей команды.

Процессор имеет 5 регистров общего назначения (РОН), непосредственно доступных программисту для написания программы. Эти регистры (A, B, C, D, E) объединены в отдельный блок и имеют идентичное устройство. Регистры выполнены на библиотечном компоненте Memory (RegistrA, RegistrB и т.д.). Компонент Memory через элемент задержки в один такт выходному значению присваивает входное. Поэтому для регистров не нужны дополнительные элементы задержки Unit Delay, как, например, для блоков Port или Stack.

Помимо пользовательских есть еще три специальных регистра (РСН). Программист не может изменять их напрямую, а только косвенно при выполнении определенных команд. Специальные регистры, как и пользовательские регистры, выполнены на компонентах Memory, которые с задержкой в один такт передают сигнал со входа на выход.

Ip — 8-разрядный регистр, использующийся для хранения текущего адреса программы. Процессор выполняет любую команду за один цикл, поэтому по умолчанию Ip увеличивается с каждым циклом синхрочастоты на единицу. При условном или безусловном переходе или при вызове подпрограммы в регистр Ip непосредственно загружается адрес следующей команды.

R — 8-разрядный регистр-указатель вершины стека. Стек имеет размер 256 ячеек, однако используется из них только 255, при этом его адресация — от 255 до 0. Указатель вершины стека инициализируется числом 255 и это символизирует то, что в стек еще ничего не было записано. Первая запись в стек производится по адресу 254, при этом указатель стека сдвигается вниз и также становится равным 254, и на выходе из стека появляется содержимое 254-й ячейки.

В процессе работы программы содержимое регистра R изменяется. Оно показывает количество свободных ячеек в стеке. При «вталкивании» числа в стек (команда PUSH в табл. 1) значение регистра R уменьшается, а при «выталкивании» из стека (команда POP) — увеличивается.

Flags — двухразрядный регистр, отображающий состояние программы. Младший разряд содержит флаг нуля (ZF), а старший — флаг переполнения (OF). Флаги вычисляются после выполнения арифметических, логических и команд сравнения. После выполнения команд переноса (MOV), команд работы с внешним портом и команд работы со стеком флаги не вычисляются. Значения флагов определяют, будет или не будет выполняться условный переход (табл. 1).

Оперативное запоминающее устройство (блок OZU, рисунок) предназначено для хранения обрабатываемых программой данных. Размер ОЗУ — 256 ячеек по 8 бит. ОЗУ имеет три входных сигнала: сигнал данных InData, адреса Addr и разрешение записи Enable и один выход данных OutData. M-файл блока ОЗУ приведен в примере 1. Чтобы входные и выходные данные не зацикливались (они образуют вместе с АЛУ цикл), добавлен элемент задержки Unit Delay 1.

```
function OutData = ozu(InData, Addr, Enable)
persistent MData;
if (isempty(MData))
    MData = zeros(1, 256, 'uint8');
end
if (Enable == 1)
    MData(Addr + 1) = InData;
end
OutData = MData(Addr + 1);
```

Пример 1. М-файл ОЗУ микропроцессорного ядра

Для описания работы ОЗУ (емкость 256 строк × 8 бит) используется полуглобальный одномерный массив MData. Массив создается при помощи встроенной функции zeros, в качестве параметров которой передается номер первого элемента массива — 1, последнего элемента — 256 и тип данных ячеек массива — uint8. При объявлении массива невозможно начать нумерацию его ячеек с нуля, поэтому при записи и чтении данных из ОЗУ приходится сдвигать адрес на единицу. При наличии сигнала Enable (режим записи) входные данные записываются по адресу Addr + 1. На выходе постоянно присутствуют выходные данные, которые считываются также по адресу Addr + 1. Для записи данных в ОЗУ используются команды MOV [xx], A; MOV [xx], B и т. д., а для чтения — MOV A, [xx]; MOV B, [xx] и т. д. (табл. 1).

Стек спроектирован на базе ОЗУ. Однако читать из стека можно только в порядке, обратном записи, и никак иначе, то есть число, записанное в стек последним, прочитается из него первым, следующим прочитается предпоследнее и т. д. На адресный вход блока стека передается значение регистра R из АЛУ, который указывает вершину стека и одновременно количество свободных ячеек в стеке. Программное описание блока стека приведено в примере 2.

```
function OutData = stack(InData, Enable, Addr)
persistent SData;
if (isempty(SData))
    SData = zeros(1, 256, 'uint8');
end
if (Enable == 1)
    SData(Addr + 1) = InData;
end
OutData = SData(Addr + 1);
```

Пример 2. М-файл стека микропроцессорного ядра

При выполнении команды CALL (запуск подпрограммы) стек используется автоматически. В стек сохраняется значение регистра счетчика команд Ip+1 (адрес, следующий за командой CALL), при этом также осуществляется сдвиг указателя вершины стека R. При возврате из подпрограммы (команда RET) из стека восстанавливается регистр Ip. Это позволяет использовать вложенные подпрограммы до 255 уровней. Программист может работать со стеком напрямую, используя команды PUSHF, POPF, PUSH A, POP A и т. п. (табл. 1).

В разрабатываемой модели процессора реализованы скалярные внешние прерывания. Для управления прерываниями имеется специальный блок (block_int). Описание блока обработки прерываний в виде М-файла приведено в примере 3.

```
function Int = block_int(FromButton, IntEnable)
if (IntEnable == 1)
    Int = uint8(FromButton);
else
    Int = uint8(0);
end
```

Пример 3. М-файл блока обработки прерываний

Если прерывания разрешены (IntEnable == 1), тогда на выход (Int) подается сигнал FromButton с внешнего устройства (Signal Builder). Разрешать или запрещать прерывания следует программно, используя команды INTON (разрешить прерывания) и INTOFF (запретить прерывания). При инициализации процессора прерывания запрещены, поэтому в начале программы нужно их разрешить. При выполнении условия elseif (Int == 1)&& (InIntEnable == 1) прерывания автоматически запрещаются командой OutIntEnable = false. Приведем фрагмент М-файла АЛУ, в котором описана данная ситуация:

```
elseif (Int == 1)&&(InIntEnable == 1) %Произошло прерывание
    OutSData = InIp;
    SEnable = true;
    OutR = InR - 1;
    OutIp = uint8(1);
    OutIntEnable = false; %Аппаратный запрет прерываний
```

Если Int = 0, значит прерывания нет, если же Int отлично от нуля, то в нем находится код прерывания. По этому коду можно определить устройство, которое вызвало прерывание.

При возникновении прерывания текущая программа останавливается, значение Ip сохраняется в стек, и процессор переходит к выполнению подпрограммы прерывания. В подпрограмме необходимо запретить прерывания и сохранить пользовательские регистры и регистр флагов в стек. Перед выходом из подпрограммы прерываний нужно восстановить регистры из стека и снова разрешить прерывания. Пример подпрограммы прерываний приведен ниже (пример 4).

```
00 JMP 14      Перепрыгиваем подпрограмму прерываний
01 INTOFF      Начинаем обработку прерывания.
               Запрещаем другие прерывания
02 PUSH A      Сохраняем содержимое регистра A в стек
03 PUSH B      Сохраняем содержимое регистра B в стек
04 PUSH C      Сохраняем содержимое регистра C в стек
05 PUSH D      Сохраняем содержимое регистра D в стек
06 PUSH E      Сохраняем содержимое регистра E в стек
07 PUSHF       Сохраняем содержимое регистра флагов в стек
08 IN A        Читаем данные из входного порта в регистр A
09 IN B        Читаем данные из входного порта в регистр B
0A XOR A,B     Исключающее ИЛИ регистров A и B,
               результат помещаем в A
0B OUT A       Выводим значение регистра A во входной порт
0C POPF        Восстанавливаем содержимое регистра флагов из стека
0D POP E       Восстанавливаем содержимое регистра E из стека
0E POP D       Восстанавливаем содержимое регистра D из стека
0F POP C       Восстанавливаем содержимое регистра C из стека
10 POP B       Восстанавливаем содержимое регистра B из стека
11 POP A       Восстанавливаем содержимое регистра A из стека
12 INTON       Снова разрешаем прерывания
13 RET         Возвращаемся из подпрограммы обработки
               прерываний
14 …
```

Пример 4. Подпрограмма прерываний

Пример 4 показывает, как должна быть оформлена подпрограмма прерываний. Начало программы должно находиться по адресу 0x01, длина программы произвольна и ограничивается только емкостью ПЗУ (256 команд).

Для взаимодействия с внешними устройствами предусмотрен порт ввода/вывода. Программное описание этого блока приведено в примере 5.

```
function OutData = port(InData, WEnable, REnable)

persistent RandomPort;
if (isempty(RandomPort))
    RandomPort = uint8(34);
end

if (REnable == 1)
    RandomPort(1) = uint8(43);
end

if (WEnable == 1)
    RandomPort(1) = InData;
end

OutData = RandomPort(1);
```

Пример 5. М-файла блока порта ввода/вывода

Для имитации работы блока порта ввода/вывода (команды IN и OUT) и блока обработки прерываний, порт инициализируется десятичным числом 34, а затем при выполнении условия REnable == 1 в порт заносится число 43. Это было сделано для того, чтобы можно было прочитать из порта два разных числа (см. подпрограмму прерываний, пример 4).

Арифметико-логическое устройство (блок ALU на рисунке) представляет собой управляющий автомат и предназначено для идентификации и выполнения команд, поступающих из ПЗУ через разделитель полей команды. Также в нем происходит обновление регистров, если они не используются в текущей команде, или их пересчет, если используются. Счетчик команд встроен в АЛУ. Пример 6 показывает М-файл арифметико-логического устройства.

```
function [OutA, OutB, OutC, OutD, OutE, OutIp, OutR, OutMData,
OutAddr,...
   MEnable, OutSData, SEnable, OutPort, OutFlags, PortWEnable, ...
   PortREnable, OutIntEnable]...
   = alu(InCmd, InData, InA, InB, InC, InD, InE, InIp, InR,
InMData,...
   InSData, InPort, Int, InFlags, InIntEnable)

% Обновление регистров
OutA = InA;
OutB = InB;
OutC = InC;
OutD = InD;
OutE = InE;
OutR = InR;
OutFlags = InFlags;
MEnable = false;
OutAddr = uint8(0);
OutMData = uint8(0);

SEnable = false;
OutSData = uint8(0);

OutPort = uint8(0);
PortWEnable = false;
PortREnable = false;

OutIntEnable = InIntEnable;

if (Int == 0)||(InIntEnable == 0) %Обработка прерываний

   OutIp = InIp + 1; %Увеличение содержимого счетчика команд
```

```
switch InCmd
  case 0 %Команды, не содержащие байт данных
    switch InData
      case 0 %NOP
      case 1 %RET
        OutIp = InSData;
        OutR = InR + 1;
      case 2 %INTON
        OutIntEnable = true;
      case 3 %INTOFF
        OutIntEnable = false;
      case 4 %PUSHF
        OutSData = uint8(InFlags);
        SEnable = true;
        OutR = InR - 1;
      case 5 %POPF
        OutFlags = fi(InSData, 0, 2, 0);
        OutR = InR + 1;
      case 6 %PUSH A
        OutSData = InA;
        SEnable = true;
        OutR = InR - 1;
      case 7 %PUSH B
        OutSData = InB;
        SEnable = true;
        OutR = InR - 1;
      case 8 %PUSH C
        OutSData = InC;
        SEnable = true;
        OutR = InR - 1;
      case 9 %PUSH D
        OutSData = InD;
        SEnable = true;
        OutR = InR - 1;
      case 10 %PUSH E
        OutSData = InE;
        SEnable = true;
        OutR = InR - 1;
      case 11 %POP A
        OutA = InSData;
        OutR = InR + 1;
      case 12 %POP B
        OutB = InSData;
        OutR = InR + 1;
      case 13 %POP C
        OutC = InSData;
        OutR = InR + 1;
      case 14 %POP D
        OutD = InSData;
        OutR = InR + 1;
      case 15 %POP E
        OutE = InSData;
        OutR = InR + 1;
      case 16 %IN A
        OutA = InPort;
        PortREnable = true;
      case 17 %IN B
        OutB = InPort;
        PortREnable = true;
      case 18 %IN C
        OutC = InPort;
        PortREnable = true;
      case 19 %IN D
        OutD = InPort;
        PortREnable = true;
      case 20 %IN E
        OutE = InPort;
        PortREnable = true;
      case 21 %OUT A
        OutPort = InA;
        PortWEnable = true;
      case 22 %OUT B
        OutPort = InB;
        PortWEnable = true;
      case 23 %OUT C
        OutPort = InC;
        PortWEnable = true;
      case 24 %OUT D
        OutPort = InD;
        PortWEnable = true;
      case 25 %OUT E
        OutPort = InE;
        PortWEnable = true;
      case 26 %DEC A
        [OutA, OutFlags] = f_sub(InA, 1);
      case 27 %DEC B
        [OutB, OutFlags] = f_sub(InB, 1);
      case 28 %DEC C
        [OutC, OutFlags] = f_sub(InC, 1);
      case 29 %DEC D
        [OutD, OutFlags] = f_sub(InD, 1);
      case 30 %DEC E
        [OutE, OutFlags] = f_sub(InE, 1);
      case 31 %INC A
        [OutA, OutFlags] = f_sum(InA, 1);
      case 32 %INC B
        [OutB, OutFlags] = f_sum(InB, 1);
      case 33 %INC C
        [OutC, OutFlags] = f_sum(InC, 1);
      case 34 %INC D
        [OutD, OutFlags] = f_sum(InD, 1);
      case 35 %INC E
        [OutE, OutFlags] = f_sum(InE, 1);
      case 36 %ADD A,B
        [OutA, OutFlags] = f_sum(InA, InB);
      case 37 %ADD A,C
        [OutA, OutFlags] = f_sum(InA, InC);
      case 38 %ADD A,D
        [OutA, OutFlags] = f_sum(InA, InD);
      case 39 %ADD A,E
        [OutA, OutFlags] = f_sum(InA, InE);
      case 40 %SUB A,B
        [OutA, OutFlags] = f_sub(InA, InB);
      case 41 %SUB A,C
        [OutA, OutFlags] = f_sub(InA, InC);
      case 42 %SUB A,D
        [OutA, OutFlags] = f_sub(InA, InD);
      case 43 %SUB A,E
        [OutA, OutFlags] = f_sub(InA, InE);
      case 44 %AND A,B
        [OutA, OutFlags] = f_and(InA, InB);
      case 45 %AND A,C
        [OutA, OutFlags] = f_and(InA, InC);
      case 46 %AND A,D
        [OutA, OutFlags] = f_and(InA, InD);
      case 47 %AND A,E
        [OutA, OutFlags] = f_and(InA, InE);
      case 48 %OR A,B
        [OutA, OutFlags] = f_or(InA, InB);
      case 49 %OR A,C
        [OutA, OutFlags] = f_or(InA, InC);
      case 50 %OR A,D
        [OutA, OutFlags] = f_or(InA, InD);
      case 51 %OR A,E
        [OutA, OutFlags] = f_or(InA, InE);
      case 52 %XOR A,B
        [OutA, OutFlags] = f_xor(InA, InB);
      case 53 %XOR A,C
        [OutA, OutFlags] = f_xor(InA, InC);
      case 54 %XOR A,D
        [OutA, OutFlags] = f_xor(InA, InD);
      case 55 %XOR A,E
        [OutA, OutFlags] = f_xor(InA, InE);
      case 56 %MOV A,B
        OutA = InB;
      case 57 %MOV A,C
        OutA = InC;
      case 58 %MOV A,D
        OutA = InD;
      case 59 %MOV A,E
        OutA = InE;
      case 60 %MOV B,A
        OutB = InA;
      case 61 %MOV B,C
        OutB = InC;
      case 62 %MOV B,D
        OutB = InD;
      case 63 %MOV B,E
        OutB = InE;
      case 64 %MOV C,A
        OutC = InA;
      case 65 %MOV C,B
        OutC = InB;
      case 66 %MOV C,D
        OutC = InD;
      case 67 %MOV C,E
        OutC = InE;
      case 68 %MOV D,A
        OutD = InA;
      case 69 %MOV D,B
        OutD = InB;
      case 70 %MOV D,C
        OutD = InC;
      case 71 %MOV D,E
        OutD = InE;
      case 72 %MOV E,A
        OutE = InA;
      case 73 %MOV E,B
        OutE = InB;
      case 74 %MOV E,C
        OutE = InC;
      case 75 %MOV E,D
        OutE = InD;
      case 76 %CMP A,B
        OutFlags = f_cmp(InA, InB);
      case 77 %CMP A,C
        OutFlags = f_cmp(InA, InC);
      case 78 %CMP A,D
        OutFlags = f_cmp(InA, InD);
      case 79 %CMP A,E
        OutFlags = f_cmp(InA, InE);
      case 80 %CMP B,A
        OutFlags = f_cmp(InB, InA);
      case 81 %CMP B,C
        OutFlags = f_cmp(InB, InC);
      case 82 %CMP B,D
        OutFlags = f_cmp(InB, InD);
      case 83 %CMP B,E
        OutFlags = f_cmp(InB, InE);
      case 84 %CMP C,A
        OutFlags = f_cmp(InC, InA);
      case 85 %CMP C,B
        OutFlags = f_cmp(InC, InB);
      case 86 %CMP C,D
        OutFlags = f_cmp(InC, InD);
      case 87 %CMP C,E
        OutFlags = f_cmp(InC, InE);
      case 88 %CMP D,A
        OutFlags = f_cmp(InD, InA);
      case 89 %CMP D,B
        OutFlags = f_cmp(InD, InB);
      case 90 %CMP D,C
        OutFlags = f_cmp(InD, InC);
      case 91 %CMP D,E
        OutFlags = f_cmp(InD, InE);
      case 92 %CMP E,A
        OutFlags = f_cmp(InE, InA);
      case 93 %CMP E,B
        OutFlags = f_cmp(InE, InB);
      case 94 %CMP E,C
        OutFlags = f_cmp(InE, InC);
      case 95 %CMP E,D
        OutFlags = f_cmp(InE, InD);
      case 96 %XCHG A,B
        X = InB;
        OutB = InA;
        OutA = X;
      case 97 %XCHG A,C
        X = InC;
        OutC = InA;
        OutA = X;
      case 98 %XCHG A,D
        X = InD;
        OutD = InA;
        OutA = X;
      case 99 %XCHG A,E
        X = InE;
        OutE = InA;
        OutA = X;
      case 100 %XCHG B,C
        X = InC;
        OutC = InB;
        OutB = X;
      case 101 %XCHG B,D
        X = InD;
        OutD = InB;
        OutB = X;
      case 102 %XCHG B,E
        X = InE;
        OutE = InB;
        OutB = X;
      case 103 %XCHG C,D
        X = InD;
        OutD = InC;
        OutC = X;
      case 104 %XCHG C,E
        X = InE;
        OutE = InC;
        OutC = X;
      case 105 %XCHG D,E
        X = InE;
        OutE = InD;
        OutD = X;
    end
  %Команды, работающие с байтом данных
  case 1 %JMP
    % Jump to new command
    OutIp = InData;
  case 2 %JZ xx
    % Jump to new command if ZF=1
    if (bitget(InFlags, 1) == 1)
      OutIp = InData;
    end
  case 3 %JNZ xx
    % Jump to new command if ZF=0
    if (bitget(InFlags, 1) ~= 1)
      OutIp = InData;
    end
  case 4 %JO xx
    % Jump to new command if ZO=1
    if (bitget(InFlags, 2) == 1)
      OutIp = InData;
    end
  case 5 %JNO xx
    % Jump to new command if ZO=0
    if (bitget(InFlags, 2) ~= 1)
      OutIp = InData;
    end
  case 6 %CALL
    OutSData = InIp + 1;
    SEnable = true;
    OutR = InR - 1;
    OutIp = InData;
  case 7 %CMP A,xx
    OutFlags = f_cmp(InA, InData);
  case 8 %CMP B,xx
    OutFlags = f_cmp(InB, InData);
  case 9 %CMP C,xx
    OutFlags = f_cmp(InC, InData);
  case 10 %CMP D,xx
```

```
        OutFlags = f_cmp(InD, InData);
    case 11 %CMP E,xx
        OutFlags = f_cmp(InE, InData);
    case 12 %MOV A,xx
        OutA = InData;
    case 13 %MOV B,xx
        OutB = InData;
    case 14 %MOV C,xx
        OutC = InData;
    case 15 %MOV D,xx
        OutD = InData;
    case 16 %MOV E,xx
        OutE = InData;
    case 17 %MOV A,[xx]
        OutAddr = InData;
        OutA = InMData;
    case 18 %MOV B,[xx]
        OutAddr = InData;
        OutB = InMData;
    case 19 %MOV C,[xx]
        OutAddr = InData;
        OutC = InMData;
    case 20 %MOV D,[xx]
        OutAddr = InData;
        OutD = InMData;
    case 21 %MOV E,[xx]
        OutAddr = InData;
        OutE = InMData;
    case 22 %MOV [xx],A
        OutAddr = InData;
        OutMData = InA;
        MEnable = true;
    case 23 %MOV [xx],B
        OutAddr = InData;
        OutMData = InB;
        MEnable = true;
    case 24 %MOV [xx],C
        OutAddr = InData;
        OutMData = InC;
        MEnable = true;
    case 25 %MOV [xx],D
        OutAddr = InData;
        OutMData = InD;
        MEnable = true;
    case 26 %MOV [xx],E
        OutAddr = InData;
        OutMData = InE;
        MEnable = true;
  end
elseif (Int == 1)&&(InIntEnable == 1) %Произошло прерывание
  OutSData = InIp;
  SEnable = true;
  OutR = InR - 1;
  OutIp = uint8(1);
  OutIntEnable = false; %Аппаратный запрет прерываний
else
  OutIp = InIp;
end

function [sum, flags] = f_sum(x1, x2)
  %Сложение с вычислением флагов
  x1_tmp = fi(x1, 0, 9, 0);
  x2_tmp = fi(x2, 0, 9, 0);
  sum_tmp = fi(x1_tmp + x2_tmp, 0, 9, 0);

  if (sum_tmp == 0)
    b_zero = fi(1, 0, 1, 0);
  else
    b_zero = fi(0, 0, 1, 0);
  end
  b_overflow = fi(bitget(sum_tmp, 9), 0, 1, 0);
  flags = fi(bitconcat(b_overflow, b_zero), 0, 2, 0);
  sum = uint8(sum_tmp);

function [sub, flags] = f_sub(x1, x2)
  %Вычитание с вычислением флагов

  x1_tmp = fi(x1, 0, 9, 0);
  x2_tmp = fi(x2, 0, 9, 0);
  sub_tmp = fi(x1_tmp - x2_tmp, 0, 9, 0);

  if (sub_tmp == 0)
    b_zero = fi(1, 0, 1, 0);
  else
    b_zero = fi(0, 0, 1, 0);
  end
  b_overflow = fi(bitget(sub_tmp, 9), 0, 1, 0);
  flags = fi(bitconcat(b_overflow, b_zero), 0, 2, 0);
  sub = uint8(sub_tmp);

function [res, flags] = f_and(x1, x2)
  %AND with flags

  res = bitand(x1, x2);
  if (res == 0)
    b_zero = fi(1, 0, 1, 0);
  else
    b_zero = fi(0, 0, 1, 0);
  end
  flags = fi(b_zero, 0, 2, 0);
```

```
function [res, flags] = f_or(x1, x2)
  %OR with flags

  res = bitor(x1, x2);
  if (res == 0)
    b_zero = fi(1, 0, 1, 0);
  else
    b_zero = fi(0, 0, 1, 0);
  end
  flags = fi(b_zero, 0, 2, 0);

function [res, flags] = f_xor(x1, x2)
  %XOR with flags

  res = bitxor(x1, x2);
  if (res == 0)
    b_zero = fi(1, 0, 1, 0);
  else
    b_zero = fi(0, 0, 1, 0);
  end
  flags = fi(b_zero, 0, 2, 0);

function flags = f_cmp(x1, x2)
  %Compare with flags

  x1_tmp = fi(x1, 0, 9, 0);
  x2_tmp = fi(x2, 0, 9, 0);
  sub_tmp = fi(x1_tmp - x2_tmp, 0, 9, 0);

  if (sub_tmp == 0)
    b_zero = fi(1, 0, 1, 0);
  else
    b_zero = fi(0, 0, 1, 0);
  end
  b_overflow = fi(bitget(sub_tmp, 9), 0, 1, 0);
  flags = fi(bitconcat(b_overflow, b_zero), 0, 2, 0);
```

Пример 6. М-файла арифметико-логического устройства

В начале листинга идет обновление регистров. Далее — оператор выбора switch, в котором перебираются все машинные инструкции и соответственно действия, выполняемые по приходу той или иной инструкции. В конце листинга — объявленные пользовательские функции для арифметического и логического вычислений и сравнения с учетом флагов. Рассмотрим одну из этих функций подробно (function [sum, flags] = f_sum (x1, x2)).

Для детектирования флага переполнения создаем временные переменные размером на один бит больше, чем входные значения (x1_tmp = fi (x1, 0, 9, 0); x2_tmp = fi (x2, 0, 9, 0);). Далее суммируем временные переменные и результат помещаем в третью временную переменную (sum_tmp = fi (x1_tmp + x2_tmp, 0, 9, 0);). sum_tmp также имеет разрядность 9 бит. Затем проверяем, не нулевой ли получился результат. Если нулевой, тогда b_zero = 1, иначе b_zero = 0, то есть получаем флаг нуля. Далее вычисляем флаг переполнения. Для этого проверяем последний, девятый бит временной переменной sum_tmp. Если он нулевой, значит переполнения нет, иначе — есть, другими словами, этот девятый бит и будет флагом переполнения. Теперь можно с помощью функции bitconcat соединить флаги в одну переменную и преобразовать сумму в привычный формат uint8. Остальные функции с вычислением флагов работают аналогичным образом.

Система MATLAB/Simulink может быть эффективно использована для ускорения процесса разработки моделей микропроцессорных ядер, однако с добавлением новых блоков и переработкой системы команд значительно усложняется управляющий автомат, на базе которого разработано АЛУ. Читателю предлагается самостоятельно, в режиме отладки, провести эксперименты с тестовой программой и подпрограммой обработки прерываний (табл. 2).

Таблица 2. Эксперимент с тестовой программой и подпрограммой обработки прерываний

Адрес	Мнемоника	HEX	DEC
Тестовая программа			
00	JMP 13	0113H	275
13	INTON	0002H	2
14	MOV A,10	0C10H	3088
15	MOV B,10	OD10H	3344
16	CMP A,B	004CH	76
17	JZ 19	0219H	537
18	MOV A,13	0C13H	3091
19	DEC A	001AH	26
1A	JNZ 19	0319H	793
1B	CMP B,10	0810H	2064
1C	JZ 1E	021EH	542
1D	MOV B,0F	0D0FH	3343
1E	MOV A,FA	0CFAH	3322
1F	ADD A,B	0024H	36
20	JO 22	0422H	1058
21	MOV B,11	0D11H	3345
22	MOV A,03	0C03H	3075
23	ADD A,B	0024H	36
24	JNO 26	0526H	1318
25	MOV A,1E	0C1EH	3102
26	MOV A,32	0C32H	3122
27	NOP	0000H	0
28	JMP 27	0127H	295
29	NOP	0000H	0
Подпрограмма обработки прерываний			
01	PUSH A	0006H	6
02	PUSH B	0007H	7
03	PUSH C	0008H	8
04	PUSH D	0009H	9
05	PUSH E	000AH	10
06	PUSHF	0004H	4
07	IN A	0010H	16
08	IN B	0011H	17
09	XOR A,B	0034H	52
0A	OUT A	0015H	21
0B	POPF	0005H	5
0C	POP E	000FH	15
0D	POP D	000EH	14
0E	POP C	000DH	13
0F	POP B	000CH	12
10	POP A	000BH	11
11	INTON	0002H	2
12	RET	0001H	1

Литература

1. Строгонов А. В., Буслов А. И. Проектирование учебного процессора для реализации в базисе ПЛИС с использованием системы MATLAB/Simulink // Компоненты и технологии. 2009. № 5.
2. Строгонов А. В. Проектирование учебного процессора с фиксированной запятой в системе Matlab/Simulink / А. В. Строгонов // Компоненты и технологии. 2009. № 7.
3. Строгонов А. В., Цыбин С. А., Буслов А. И. Проектирование микропроцессорных ядер с использованием приложения StateFlow системы MATLAB/Simulink // Компоненты и технологии. 2010. № 1.
4. Тарасов И. Проектирование конфигурируемых процессоров на базе ПЛИС. Часть II // Компоненты и технологии. 2006. № 3.