

Программный секвенсор ADSP-21060

1. Функции секвенсора
2. Архитектура секвенсора. Этапы выполнения инструкции
3. Переходы, вызовы подпрограмм, возвраты
4. Организация циклов
 1. Стеки, используемые при организации циклов
 2. Циклы по счетчику
 3. Циклы по арифметическому условию
 4. Ограничения при организации циклов
 5. Флаги состояния секвенсора
5. Кэш инструкций
 1. Архитектура кэша
 2. Пример влияния размещения сегментов в памяти на эффективность выполнения программы

"Работа выполнена в рамках реализации "Программы развития ФГОУ ВПО "Южный федеральный университет" на 2007-2010гг. ("Разработка образовательных контентов и ресурсов нового поколения").

Хусаинов Н.Ш.

Технологический институт Южного федерального университета в г.Таганроге, 2007г.

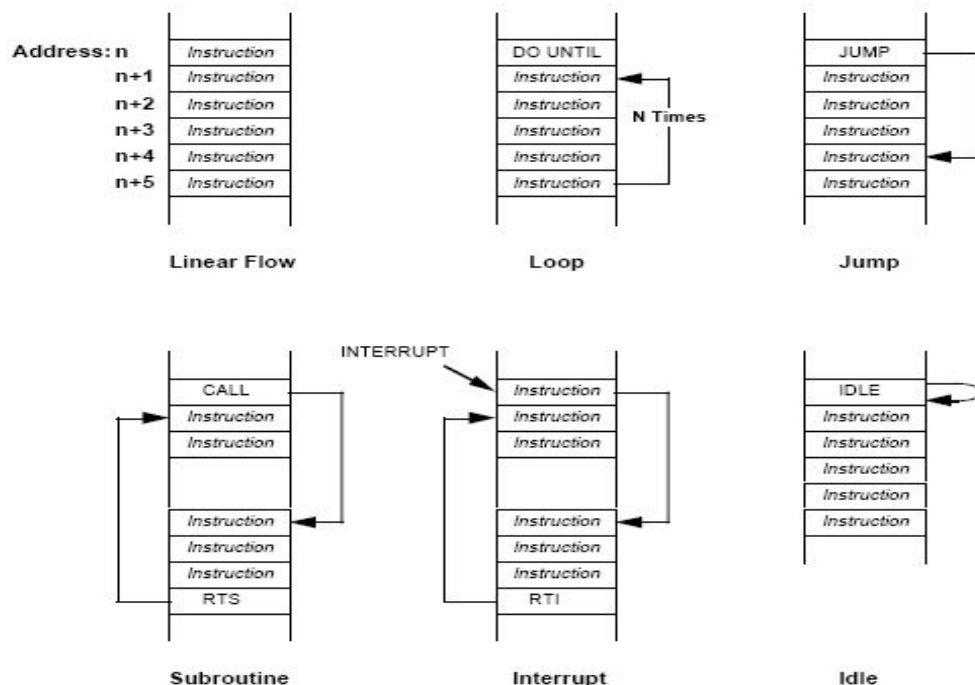
Функции секвенсора

Выполнение инструкций в линейном порядке с инкрементом адреса выборки следующей команды.

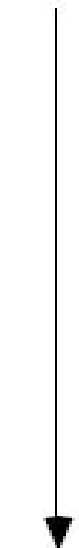
Варианты нарушения линейного порядка следования инструкций:

- переходы (условные и безусловные);
- вызовы подпрограмм и возвраты (условные и безусловные);
- вызовы обработчиков прерываний и возвраты (условные и безусловные);

- циклы;
- инструкция Idle.



Архитектура секвенсора. Этапы выполнения инструкции



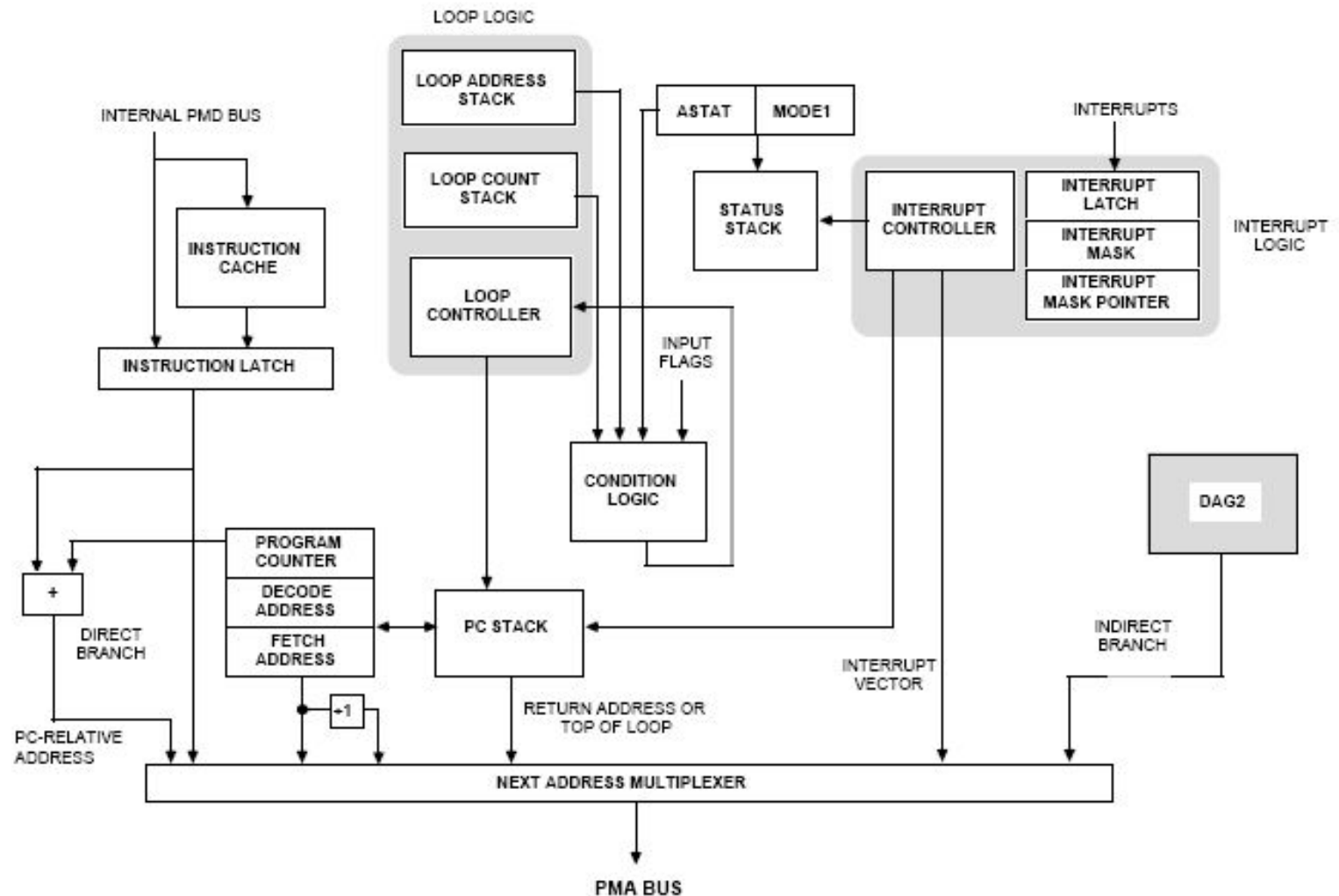
time (cycles)	Fetch	Decode	Execute
1	0x08		
2	0x09	0x08	
3	0x0A	0x09	0x08
4	0x0B	0x0A	0x09
5	0x0C	0x0B	0x0A

Адрес выполняемой (Execute) инструкции – регистр PC.

Адрес декодируемой (Decode) инструкции – регистр DADDR.

Адрес выбираемой (Fetch) инструкции – регистр FADDR.

Архитектура секвенсора. Этапы выполнения инструкции



Архитектура секвенсора. Этапы выполнения инструкции

Регистры программного секвенсора

Регистр	Содержимое	Разрядность	Прим.
FADDR	Адрес выбираемой из памяти инструкции	24	Только чтение
DADDR	Адрес декодируемой инструкции	24	Только чтение
PC	Адрес выполняемой инструкции	24	Только чтение
PCSTK	Содержимое верхней ячейки стека PC	24	
PCSTKP	Указатель стека PC	5	
LADDR	Содержимое верхней ячейки стека цикла	32	
CURLCNT R	Содержимое верхней ячейки стека счетчика цикла	32	
LCNTR	Число итераций следующего цикла	32	

Переходы и вызовы подпрограмм

Переходы, вызовы подпрограмм, возвраты

При выполнении переходов, вызовов и возвратов выборка очередной инструкции выполняется по адресу, который не является следующим по счету за адресом предыдущей выборки.

При вызове подпрограммы дополнительно в стек Программного секвенсора (PC Stack) помещается адрес возврата.

При возврате из подпрограммы или из обработчика прерывания адрес возврата выталкивается из стека Программного секвенсора (при возврате из обработчика прерывания выполняются и дополнительные действия).

Переходы и вызовы подпрограмм

Варианты выполнения

Переходы, вызовы и возвраты могут быть условными и безусловными:

```
IF cond jump Proc2;      jump Proc2;
```

При переходах и вызовах режим адресации может быть прямой, косвенный, относительный:

```
call Proc3;  
IF cond call (M12, I14);  
jump (PC, 0x0004);
```

Переходы, вызовы и возвраты могут быть обычные и задержанные:

```
call Proc4;              rts (db);
```

Переход может прерывать выполнение цикла.

```
jump Proc5 (I1);
```

Переходы и вызовы подпрограмм

Обычный («незадержанный») переход (вызов, возврат)

NON-DELAYED JUMP OR CALL

CLOCK CYCLES →

Execute Instruction	n	nop	nop	j
Decode Instruction	n+1->nop	n+2->nop	j	j+1
Fetch Instruction	n+2	j	j+1	j+2

n+1 suppressed

n+2 suppressed; for
call, n+1 pushed on
PC stack

n = Branch instruction

j = Instruction at Jump or Call address

r = Instruction at Return address

NON-DELAYED RETURN

CLOCK CYCLES →

Execute Instruction	n	nop	nop	r
Decode Instruction	n+1->nop	n+2->nop	r	r+1
Fetch Instruction	n+2	r	r+1	r+2

n+1 suppressed

n+2 suppressed; r
popped from PC
stack

Недостаток – необходимость перезагрузки конвейера и потеря двух тактов.

Преимущество – понятность: порядок выполнения соответствует порядку записи.

Переходы и вызовы подпрограмм

Задержанный переход (вызов, возврат)

DELAYED JUMP OR CALL

CLOCK CYCLES →

Execute Instruction	n	n+1	n+2	j
Decode Instruction	n+1	n+2	j	j+1
Fetch Instruction	n+2	j	j+1	j+2

for call, n+3
pushed on PC
stack

n = Branch instruction

j = Instruction at Jump or Call address

r = Instruction at Return address

DELAYED RETURN

CLOCK CYCLES →

Execute Instruction	n	n+1	n+2	r
Decode Instruction	n+1	n+2	r	r+1
Fetch Instruction	n+2	r	r+1	r+2

r popped from
PC stack

Преимущество – отсутствует необходимость перезагрузки конвейера.
Недостаток – усложнение программы.

Ограничение: в 2-х адресах за инструкцией задержанного перехода не могут находиться инструкции других переходов (вызовов, возвратов), команды работы со стеком Программного секвенсора (PC Stack), инструкции организации циклов (DO... UNTIL...), команда IDLE.

Переходы и вызовы подпрограмм

Мнемоники условных инструкций

<u>No.</u>	<u>Mnemonic</u>	<u>Description</u>	<u>True If</u>
0	EQ	ALU equal zero	AZ = 1
1	LT	ALU less than zero	See Note 1 below
2	LE	ALU less than or equal zero	See Note 2 below
3	AC	ALU carry	AC = 1
4	AV	ALU overflow	AV = 1
5	MV	Multiplier overflow	MV = 1
6	MS	Multiplier sign	MN = 1
7	SV	Shifter overflow	SV = 1
8	SZ	Shifter zero	SZ = 1
9	FLAG0_IN	Flag 0 input	FI0 = 1
10	FLAG1_IN	Flag 1 input	FI1 = 1
11	FLAG2_IN	Flag 2 input	FI2 = 1
12	FLAG3_IN	Flag 3 input	FI3 = 1
13	TF	Bit test flag	BTF = 1
14	BM	Bus Master	
15	LCE	Loop counter expired (DO UNTIL term)	CURLCNTR = 1
15	NOT LCE	Loop counter not expired (IF cond)	CURLCNTR ≠ 1
<i>Bits 16-30 are the complements of bits 0-14</i>			
16	NE	ALU not equal to zero	AZ = 0
17	GE	ALU greater than or equal zero	See Note 3 below
18	GT	ALU greater than zero	See Note 4 below
19	NOT AC	Not ALU carry	AC = 0
20	NOT AV	Not ALU overflow	AV = 0
21	NOT MV	Not multiplier overflow	MV = 0
22	NOT MS	Not multiplier sign	MN = 0
23	NOT SV	Not shifter overflow	SV = 0
24	NOT SZ	Not shifter zero	SZ = 0
25	NOT FLAG0_IN	Not Flag 0 input	FI0 = 0
26	NOT FLAG1_IN	Not Flag 1 input	FI1 = 0
27	NOT FLAG2_IN	Not Flag 2 input	FI2 = 0
28	NOT FLAG3_IN	Not Flag 3 input	FI3 = 0
29	NOT TF	Not bit test flag	BTF = 0
30	NBM	Not Bus Master	
31	FOREVER	Always False (DO UNTIL)	always
31	TRUE	Always True (IF)	always

Переходы и вызовы подпрограмм

Примеры использования условных инструкций

Условный переход (вызов, возврат):

```
IF EQ jump Proc2;  
IF NE call Proc2 (db);  
IF NOT SZ rts;
```

Условное выполнение :

```
IF LE R1=R1+R4;  
IF GT F4=F8*F9, R2=dm(I7,m0);  
IF TF call (M8,I15), ELSE R2=R2+R8;
```

Всегда истинное условие:

```
IF TRUE call Proc4;
```

Бесконечный цикл (всегда ложное условие):

```
DO Label UNTIL FOREVER;
```

Организация циклов

Общие сведения

Процессор SHARC ADSP имеет механизм аппаратной поддержки циклов.

Проверка условия выхода из цикла проверяется за 3 такта до конца тела цикла, что позволяет избежать неверного выбора Fetch-адреса и дополнительных циклов для перезагрузки конвейера (для циклов длиной из 3-х и более инструкций).

Использование специализированных стеков Программного секвенсора позволяет использовать аппаратную поддержку циклов до 6 уровней вложенности.

Есть ограничения на использование команд переходов, вызовов и возвратов в конце тела цикла.

Особого внимания требует организация коротких циклов с выходом по арифметическому условию.

Стеки, используемые при организации циклов

Занесение значений в вершины стеков выполняется инструкцией

```
DO LabelEndLoop UNTIL cond;  
...  
...  
...  
LabelEndLoop: ...
```

Выталкивание значений из вершин стеков выполняется автоматически при проверке условия выхода (если оно истинно) или при выполнении инструкции перехода с прерыванием цикла

```
jump LabelOutOfLoop (LA);
```

Стек	Наименование	Назначение (при организации циклов)
PC Stack	Стек Программного секвенсора	Адрес первой инструкции тела цикла
Loop Address Stack	Стек адреса цикла	Адрес последней инструкции цикла, код завершения, тип цикла
Loop Counter Stack	Стек счетчика цикла	Количество итераций до конца цикла

Стеки, используемые при организации циклов

Стек Программного секвенсора (PC Stack)

Глубина – 30 слов.

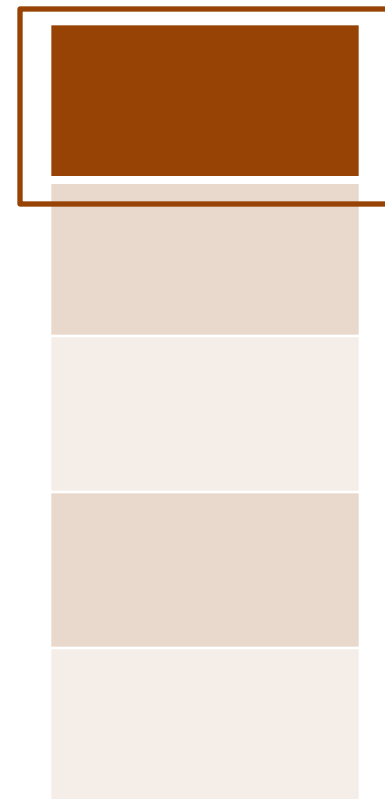
Разрядность – 24 бита.

Регистр PCSTK – содержит верхушку стека.

Указатель PCSTKP – содержит количество занятых ячеек в PC Stack.

PCSTKP=0, если стек пуст.

PCSTK



Принудительные запись/выталкивание

PUSH PCSTK; POP PCSTK;

PCSTK = 0xUUUU UUUU;

Стеки, используемые при организации циклов

Стек адреса цикла (Loop Address Stack)

Глубина – 6 слов.

Разрядность – 32 бита.

Регистр LADDR – содержит верхушку стека.

Если LADDR=0xFFFF FFFF, то стек адреса цикла пуст.

LADDR

31	30	28	24	23	0
Код типа цикла:		Код завершения		Адрес последней инструкции тела цикла	
«00»-арифм. «01» – по счетчику, 1 инструкция «02» – по счетчику, 2 инструкции «03» - по счетчику, 3 и более инструкций		Соответствует условию после UNTIL			

Стеки, используемые при организации циклов

Стек счетчика цикла (Loop Counter Stack)

Глубина – 6 слов.

Разрядность – 32 бита.

Регистр CURLCNTR – содержит верхушку стека.

Если CURLCNTR=0xFFFF FFFF, то стек счетчика цикла пуст.

Регистр LCNTR находится «над вершиной» стека счетчика цикла.

При выполнении команды Do...UNTIL его значение проталкивается в

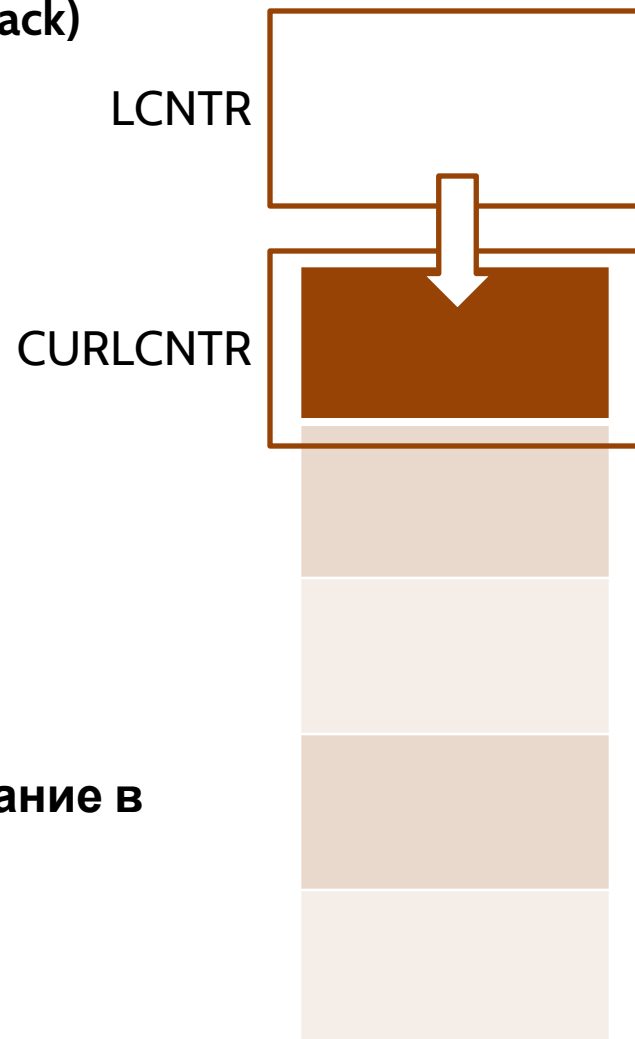
Стеки
Принудительные запись/выталкивание в стеки адреса и счетчика циклов

```
LCNTR = value;
```

```
POP LOOP;
```

```
LADDR = struct;
```

```
PUSH LOOP;
```



Циклы по счетчику

Цикл по счетчику длиной 3 и более инструкций

0x100: ...

0x101: LCNTR = 2, DO EndLoop UNTIL LCE;

0x102: R1 = R1 + 1;

0x103: R2 = R2 - 1;

Проверка условия
выхода из цикла

0x104: DM(I3,M8) = R1;

0x105: EndLoop: DM(I4,M1) = R2;

0x106: ...

PC	0x101	0x102	0x103	0x104	0x105	0x102	0x103	0x104	0x105	0x106
DADDR	0x102	0x103	0x104	0x105	0x102	0x103	0x104	0x105	0x106	0x107
FADDR	0x103	0x104	0x105	0x102	0x103	0x104	0x105	0x106	0x107	0x108
	->PCSTK ->LAS ->SCS		CURLCNTR = 1 ? CURLCNTR--	Нет			CURLCNT = 1 ? CURLCNTR— PCSTK-> LAS-> SCS->	Да		

Циклы по счетчику

Цикл по счетчику длиной в 1 инструкцию из 3-х и более итераций

```

0x100:      ...
0x101:      LCNTR = 5, DO EndLoop UNTIL LCE;
0x102: EndLoop:  DM(I4,M1) = R2;
0x103:      ...
    
```

Проверка условия
выхода из цикла

PC	0x101	0x102	0x102	0x102	0x102	0x102	0x103
DADDR	0x102	0x102	0x102	0x102	0x102	0x103	0x104
FADDR	0x103	0x102	0x102	0x102	0x103	0x104	0x105
	->PCSTK ->LAS ->SCS	CURLCNTR = 3 ? CU RLCNTR-- (=4)	CURLCNTR = 3 ? CURLCNTR-- (=3)	CURLCNT = 3 ? CURLCNTR=0xffffffff PCSTK->; LAS->; SCS->			

Циклы по счетчику

Цикл по счетчику длиной в 1 инструкцию из 1-й итерации

0x101: LCNTR = 1, DO EndLoop UNTIL LCE;

PC	0x101	0x102	nop	nop	0x103
DADDR	0x102	0x102	0x102	0x103	0x104
FADDR	0x103	0x102	0x103	0x104	0x105
	->PCSTK ->LAS ->SCS	CURLCNTR = 1 ? CU RLCNTR=0xffffffff PCSTK->; LAS->; SCS->			

Цикл по счетчику длиной в 1 инструкцию из 2-х итераций

0x101: LCNTR = 2, DO EndLoop UNTIL LCE;

PC	0x101	0x102	0x102	nop	nop	0x103
DADDR	0x102	0x102	0x102	0x102	0x103	0x104
FADDR	0x103	0x102	0x102	0x103	0x104	0x105
	->PCSTK ->LAS ->SCS	CURLCNTR = 1 ? CU RLCNTR-- (=1)	CURLCNTR = 1 ? CU RLCNTR=0xffffffff PCSTK->; LAS->; SCS->			

Цикл по счетчику из одной инструкции должен выполняться не менее 3-х раз чтобы избежать потерь производительности

Циклы по счетчику

Цикл по счетчику длиной в 2 инструкцию из 2-х и более итераций

```

0x100:      ...
0x101:      LCNTR = 3, DO EndLoop UNTIL LCE;
0x102:      DM(I4,M1) = R1;
0x103: EndLoop:  DM(I4,M1) = R2;
0x104:      ...
    
```

Проверка условия
выхода из цикла

PC	0x101	0x102	0x103	0x102	0x103	0x102	0x103	0x104
DADDR	0x102	0x103	0x102	0x103	0x102	0x103	0x104	0x105
FADDR	0x103	0x102	0x103	0x102	0x103	0x104	0x105	0x106
	->PCSTK ->LAS ->SCS		CURLCNTR = 2 ? CURLCNTR- (=2)		CURLCNT = 2 ? CURLCNTR = 0xFFFFFFFF PCSTK->; LAS->; SCS->			

Циклы по счетчику

Цикл по счетчику длиной в 2 инструкции из 1-й итерации

```
0x100:      ...
0x101:      LCNTR = 1, DO EndLoop UNTIL LCE;
0x102:      DM(I4,M1) = R1;
0x103: EndLoop:  DM(I4,M1) = R2;
0x104:      ...
```

PC	0x101	0x102	0x103	nop	nop	0x104
DADDR	0x102	0x103	0x102	0x103	0x104	0x105
FADDR	0x103	0x102	0x103	0x104	0x105	0x106
	->PCSTK ->LAS ->SCS		CURLCNTR = 1 ? CU RLCNTR=0xffffffff PCSTK->; LAS->; SCS->			

Цикл по счетчику из двух инструкций должен выполняться не менее 2-х раз чтобы избежать потерь производительности

Циклы по счетчику

Примеры неоптимальной и оптимальной организации цикла по счетчику

Задание.

Даны массивы A и B одинаковой размерности (N элементов).

Вычислить:

Неоптимальная организация цикла - 1

```
IO = A;  M0 = 1;  I8 = B;  M8 = 1;
```

```
...
```

```
MRF = 0;
```

```
R1 = 0;
```

```
R2 = 0;
```

```
LCNTR = N, DO LabelEndLoop UNTIL LCE;
```

```
    R1 = DM(IO,M0), R2 = PM(I8,M8);
```

```
LabelEndLoop:    MRF = MRF + R1*R2 (SSI);
```

```
    R0 = MR0F;
```

Количество тактов: $3*1 + 1*1 + (2*2 + (N-2)*1 + N*1) + 1*1 = 2*N+7$

Выполняется «лишнее» чтение из памяти с дополнительным сдвигом указателей

Циклы по счетчику

Примеры неоптимальной и оптимальной организации цикла по счетчику

Неоптимальная организация цикла - 2

```
I0 = A;  M0 = 1;  I8 = B;  M8 = 1;
```

```
...
```

```
MRF = 0;
```

```
R1 = 0;
```

```
R2 = 0;
```

```
LCNTR = N+1, DO LabelEndLoop UNTIL LCE;
```

```
LabelEndLoop:      MRF = MRF + R1*R2 (SSI), R1 = DM(I0,M0), R2 = PM(I8,M8);
```

```
R0 = MRF;
```

Количество тактов: $3*1 + 1*1 + (3*2 + (N-2)*1) + 1*1 = N+9$

Выполняется «лишнее» чтение из памяти с дополнительным сдвигом указателей

Циклы по счетчику

Примеры неоптимальной и оптимальной организации цикла по счетчику

Оптимальная организация цикла

```
I0 = A;  M0 = 1;  I8 = B;  M8 = 1;
```

```
...  
MRF = 0, R1 = DM(I0,M0), R2 = DM(I8,M8);
```

```
LCNTR = N-1, DO LabelEndLoop UNTIL LCE;
```

```
LabelEndLoop:    MRF = MRF + R1*R2 (SSI), R1 = DM(I0,M0), R2 = PM(I8,M8);
```

```
    R0 = MRF + R1 * R2 (SSI);
```

Количество тактов: $1*2 + 1*1 + (2*2 + (N-3)*1) + 1*1 = N+5$

«Лишнее» чтение из памяти с дополнительным сдвигом указателей не выполняется

Циклы по счетчику

Примеры неоптимальной и оптимальной организации цикла по счетчику
Задание.

Даны массивы вещественных чисел A и B одинаковой размерности (N элементов).

Вычислить:

Неоптимальная организация цикла

```
IO = A;  M0 = 1;  I8 = B;  M8 = 1;
```

```
...
```

```
F8 = 0;
```

```
LCNTR = N, DO LabelEndLoop UNTIL LCE;
```

```
    R1 = DM(IO,M0), R2 = PM(I8,M8);
```

```
    F3 = F1 * F2;
```

```
LabelEndLoop:    F8 = F8 + F3;
```

```
    DM(...) = R8;
```

Количество тактов: $1*1 + 1*1 + (1*2 + (N-1)*1 + 2*N*1) + 1*1 = 3*N+4$

Циклы по счетчику

Примеры неоптимальной и оптимальной организации цикла по счетчику

Оптимальная организация цикла

```
I0 = A;  M0 = 1;  I8 = B;  M8 = 1;
```

```
...
```

```
R1 = DM(I0,M0), R4 = PM(I8,M8);
```

```
F8 = F1*F4, R1 = DM(I0,M0), R4 = PM(I8,M8);
```

```
F12 = F1*F4, R1 = DM(I0,M0), R4 = PM(I8,M8);
```

```
LCNTR = N-3, DO LabelEndLoop UNTIL LCE;
```

```
LabelEndLoop:      F12 = F1*F4, F8 = F8+F12, R1 = DM(I0,M0), R4 =  
PM(I8,M8);
```

```
F12 = F1*F4, F8 = F8+F12;
```

```
F8 = F8 + F12;
```

```
DM(...) = R8;
```

Количество тактов: $1*2 + 1*2 + 1*2 + 1*1 + (3*2 + (N-6)*1) + 1*1 + 1*1 + 1*1 = N+10$

Циклы по арифметическому условию

Цикл по арифм. условию длиной 3 и более инструкций

```
0x100:      R1 = 2;  
0x101:      DO EndLoop UNTIL EQ;  
0x102:      R1 = R1 - 1;  
0x103:      R2 = R2 + R6;  
0x104:      DM(I3,M8) = R1;  
0x105: EndLoop:  DM(I4,M1) = R2;  
0x106:      ...
```

— | Проверка условия
выхода из цикла

PC	0x101	0x102	0x103	0x104	0x105	0x102	0x103	0x104	0x105	0x106
DADDR	0x102	0x103	0x104	0x105	0x102	0x103	0x104	0x105	0x106	0x107
FADDR	0x103	0x104	0x105	0x102	0x103	0x104	0x105	0x106	0x107	0x108
	->PCSTK ->LAS ->SCS		AZ ?	Нет			AZ ? PCSTK->; LAS->; SCS->	Да		

Циклы по арифметическому условию

Сложности в организации короткого (1-2 инструкции) цикла по арифметическому условию - 1

```
0x100:      R1 = 3;  
0x101:      DO EndLoop UNTIL EQ;  
0x102:  EndLoop:  R1 = R1 - 1;  
0x103:      ...
```

Проверка условия
выхода из цикла

PC	0x101	0x102	0x102	0x102	0x102	0x102	0x102	0x103
DADDR	0x102	0x102	0x102	0x102	0x102	0x102	0x103	0x104
FADDR	0x103	0x102	0x102	0x102	0x102	0x103	0x104	0x105
	->PCSTK ->LAS ->SCS	AZ ? (R1=2)	AZ ? (R1=1)	AZ ? (R1=0)	AZ ! (R1=-1) PCSTK->; LAS->; SCS->	(R1=-2)	(R1=-3)	

Условие выхода из цикла проверяется в начале процессорного такта.

Значение флага AZ устанавливается по результатам выполнения операции АЛУ в конце процессорного такта.

Выполняются дополнительные итерации цикла.

Циклы по арифметическому условию

Сложности в организации короткого (1-2 инструкции) цикла по арифметическому условию - 2

```

0x100:      R1 = 2;
0x101:      DO EndLoop UNTIL EQ;
0x102:      R5 = DM(I1,M2);
0x103: EndLoop: R1 = R1 - 1;
0x104:      ...
    
```

Проверка условия
выхода из цикла

PC	0x101	0x102	0x103	0x102	0x103	0x102	0x103	0x102	0x103	0x104
DADDR	0x102	0x103	0x102	0x103	0x102	0x102	0x102	0x103	0x104	0x105
FADDR	0x103	0x102	0x103	0x102	0x103	0x103	0x103	0x104	0x105	0x106
	->PCSTK ->LAS ->SCS		AZ ? (R1=1)	AZ-const	AZ ? (R1=0)	AZ-const	AZ ? (R1=-1) PCSTK->; LAS->; SCS->		(R1=-2)	

Условие выхода из цикла проверяется в начале процессорного такта.

Значение флага AZ устанавливается по результатам выполнения операции АЛУ в конце процессорного такта.

Выполняются дополнительные итерации цикла.

Циклы по арифметическому условию

Сложности в организации короткого (1-2 инструкции) цикла по арифметическому условию - 3

```
0x100:      R1 = 2; R5 = 6;
0x101:      DO EndLoop UNTIL EQ;
0x102:      R5 = R5 + 1;
0x103: EndLoop: R1 = R1 - 1;
0x104:      ...
```

Проверка условия
выхода из цикла

PC	0x101	0x102	0x103	0x102	0x103	0x102	0x103	0x102	0x103	0x102
DADDR	0x102	0x103	0x102	0x103	0x102	0x102	0x102	0x103	0x102	0x103
FADDR	0x103	0x102	0x103	0x102	0x103	0x103	0x103	0x102	0x103	0x102
	->PCSTK ->LAS ->SCS		AZ ? (R1=1) AZ = 0	AZ = 0	AZ ? (R1=0) AZ = 1	AZ = 0	AZ ? (R1=-1) AZ = 0	AZ = 0	AZ ? (R1=-2) AZ = 0	AZ = 0

Значение флага AZ, проверяемого в начале цикла выполнения инструкции 0x103, изменяется инструкцией 0x102.

Возможно получение бесконечного цикла.



Ограничения при организации циклов

Вложенные циклы не могут заканчиваться на одной и той же инструкции.

Для предотвращения потерь производительности циклы по счетчику из одной инструкции должны иметь не менее 3-х итераций, а из двух инструкций – не менее 2-х итераций.

«Короткие» циклы не должны содержать никакие команды перехода, вызова и возврата.

Последние 3 инструкции любого цикла не должны содержать никакие команды перехода, вызова и возврата. В противном случае цикл может быть отработан некорректно. Единственное исключение – «незадержанный» вызов подпрограммы CALL и возврат RTS с модификатором (LR).

Флаги состояния секвенсора

Флаги состояния стеков Программного секвенсора в регистре STKY

Флаг	Описание	Когда устанавливается	Прим.
PCFL	Стек PC Stack полон	Когда в стек записывается слово № 29 (из 30-х возможных)	не липкий
PCEM	Стек PC Stack пуст		не липкий
SSOV	Переполнение стека статуса	Когда заносится слово в полный стек	липкий
SSEM	Стек статуса пуст		не липкий
LSOV	Переполнение стека адреса и счетчика цикла	Когда заносится слово в полный стек	липкий
LSEM	Стеки адреса и счетчика цикла пусты		не липкий

Кэш инструкций

Принцип работы кэша

Кэш инструкций – ассоциативный кэш с местом для хранения 32-х инструкций, «прозрачный» для программиста.

Процессор кэширует только те инструкции, выборка которых из РМ-памяти конфликтует с обращением к данным в РМ:

PC => 0x101: R1 = R2 + R3, R2 = РМ(18,19) ;

Decode => 0x102: R4 = 0 ;

Fetch => 0x103: R5 = R2*R4 ;

При выборке инструкции процессор всегда сначала обращается за инструкцией в кэш, а при кэш-промахе – в РМ-память, вызывая дополнительный цикл на выборку инструкции.

Архитектура кэша

	LRU Bit	Instruction	Address	Valid Bit
Set 0	<input type="checkbox"/>			
Set 1	<input type="checkbox"/>			
Set 2	<input type="checkbox"/>			
•	•	•	•	•
•	•	•	•	•
Set 13	<input type="checkbox"/>			
Set 14	<input type="checkbox"/>			
Set 15	<input type="checkbox"/>			

Выбор набора осуществляется по младшим 4 битам адреса инструкции. В поле Address хранятся только старшие 20 битов адреса инструкции.

Бит LRU (Least Recently Used) – показывает более редко используемую инструкции в наборе (кандидат на замещение новой инструкцией).

Бит Valid Bit показывает наличие реальных данных в каждой строке набора.

Пример влияния размещения сегментов в памяти на эффективность выполнения программы

```
0x0100:  LCNTR=1024, DO tight UNTIL LCE;
0x0101:      R0=DM(I0,M0),  PM(I9,M9);
0x0102:      R1 = R0-R15;
0x0103:      IF EQ CALL sub;
0x0104:      F2 = FLOAT R1;
0x0105:      F3 = F2 * F2;
0x0106:  tight:  F3 = F3 + F4;
...
0x0200:  sub: R1 = R13;
0x0201:      R14 = PM(I9,M9);
0x0202:      R6 = R6 + 1;
0x0203:      R7 = PASS R12;
...
0x0211:      PM(I9,M9) = R12;
0x0212:      R4 = R4-1;
0x0213:      R7 = R5+R6;
...
0x021F:      rts;
```

Три инструкции, выполняющие обращение к РМ-памяти и вызывающие конфликт доступа к памяти. Кэшируемые инструкции (0хУУУ3), попадают в один набор (№ 3) кэша, что приводит к постоянному замещению самой «старой» инструкции на новую.

Как следствие – кэш неэффективен.

Решение – сдвинуть сегменты в памяти друг относительно друга, чтобы «проблемные» инструкции приходились на разные наборы кэша.