ST10 Family

Arquitetura

A família ST10 é composta por unidades de operações lógicas de valores de ponto flutuante e inteiro. A primeiro (MCU) é uma complexa unidade destinada a estender sinais; a segunda é a ULA. Como adaptação, as instruções que originalmente são compostas por 16 bits, que seriam estendidas a 32 bits de acordo com a necessidade de cada operação, fixou-se 32 bits em todas as instruções da ROM, preenchendo-se os 16 bits iniciais com 0, caso não utilizados.

Por se tratar de uma arquitetura com traços RISC e CISC, as operações podem ser escritas diretamente na RAM, dessa forma, não há operações direcionadas a *load* e *store*.

Não há, também, um registrador com valor constante ou acumulador.

No processador da equipe, a ULA sempre realiza operações, pois não possui um *enable*. Entretanto, o resultado das operações só será salvo quando o *enable* da RAM ou do Banco de Registradores esteja ativado.

Como os 16 primeiros bits destinado ao endereçamento da RAM sempre estarão preenchidos, a cada instrução da ROM, a RAM terá em sua saída um dado, por causa de sua leitura ser assíncrona.

As operações possuem apenas dois operando envolvidos, sendo que o primeiro deles sempre recebe o resultado da operação. Ex.: ADD R1, R2 (A operação de soma de R1 com R2 será armazenada em R1).

Operações lógicas, aritméticas e de comparação só recebem valores indiretos do segundo operando e nos registradores de 0 a 3.

Mudanças

O ST10 não possui um registrador com valor constante, portanto, alterou-se o R0 para receber dados. Desse modo, há uma operação na ULA para mover dados, o “MOV”. Há dois tipos dessa operação, uma que envolve mover um valor imediato para um registrador e outra para mover dados entre registradores.

A fim de que a operação de MOV ocorra corretamente, é preciso desconsiderar a entrada SrcA da ULA, pois quando esta recebe um valor da RAM sem que o endereço selecionado contenha um dado, ao realizar uma operação tal como

“SrcA-SrcA+SrcB” para este propósito, a saída da ULA fica com valor X”UUUU”, que não é booleano.

A operação CMP, no ST10 original, realiza uma soma com o complemento de dois do operando SrcB. Como a arquitetura projetada pela equipe está disposta a sempre reescrever o registrador utilizado em SrcA, o valor contido nele seria alterado. E como ocorre o mesmo problema que em MOV, a operação até então utilizada “SrcB-SrcB+SrcA” foi modificada para “SrcA”. Assim, o valor de SrcA se mantém e a comparação é feita pelos flags da ULA durante essa instrução da ROM.

Para as operações presentes nesta ULA, não é necessário o uso de sinal estendido, pois na Instr há bits para comportar os 16 bits que entram como SignImm, dado coletado do endereço da ROM.

O Jump é uma operação que o processador executa, dessa forma, recebe os 8 primeiro bits do Op, necessitando de um ciclo de execução, e portanto, nessa Instr, não há uma operação efetiva que a ULA poderia executar.

Alterou-se a saída do PC para 8 bits, que selecionam 256 Instr presentes na ROM. As Instrs, por sua vez, são compostas de 32 bits, sendo sempre os 8 primeiros direcionadores de Op. Ademais, esta família de processadores possui apenas um formato de função, que é ajustada de acordo com a Op a ser executada.

O ST10 tem 16 registradores word e 16 registradores byte, mas manteve-se os 8 do processador genérico como word.

Codificação e Assembly

Mudanças

A Op de ADD entre dois registradores é, originalmente, X”00”, entretanto, causa conflito com o “else X”00”” do mux do ALUControl, desta forma, alterou-se para X”01”.

As operações que não utilizam os Instr (15 downto 0) bits foram preenchidas com 0.

O Jump JMPS é um pulo incondicional para valores absolutos. Apesar disso, para as operações utilizadas no processador, em nenhum momento foi necessário pular, além do segmento do PC, o endereço da RAM. Dessa forma, os 16 bits iniciais (MM MM (Instr (15 downto 0)) ficam nulos e não são utilizados em nenhum momento.

O Jump JMPR é um salto condicional, ou seja, de valor relativo. Realiza-se uma operação de soma entre a posição atual do PC com o valor em complemento de dois contido na Instr atual da ROM.

Nas operações de SUB, MOV, CMP, AND e ADD, que se utilizam de valor imediato, há 4 bits desnecessários para o registrador RD2 (R**R**). Manteve-se o padrão dos primeiros 4 bits mais significativos para RD1 (Instr (23 downto 20)) e não utilizou-se do **R** (Instr (19 downto 16)).

No ST10 original, a RAM contém 16 bits de combinação de endereçamento (MM MM). Essa característica foi mantida, mas alterou-se para 256 endereços, já que não há necessidade de mais memória. Os pacotes de endereçamento se dão de 2 em 2 bytes, formando 16 bits. No código da equipe, o offset é realizado somando-se uma unidade, já que a estrutura da RAM no Quartus é distinta. Enquanto que no ST10 original os endereços da RAM são dispostos de dois em dois, no projeto em questão, são dispostos de um em um.

A operação MOV de opcode B8 tem formato diferenciado em relação às demais operações que contém valores indiretos em seus registradores. Na posição de RD1, o registrador, na realidade, seria RD2, pois o endereço contido em RD2 receberia o valor da saída da ULA. Para manter o padrão do código, RD1 foi invertido com RD2.

Instruções utilizadas no Processador

Mnemonic	Format
ADD Rwn, Rwm	01 nm
ADD reg, #data16	06 nm
ADD reg, mem	02 RR MM MM
ADD mem, reg	04 RR MM MM
ADD Rwn, [Rwi]	08 n:10ii
ADD Rwn, [Rwi+]	08 n:11ii
AND Rwn, Rwm	60 nm
AND reg, #data16	66 RR ## ##
AND reg, mem	62 RR MM MM
AND mem, reg	64 RR MM MM
AND Rwn, [Rwi]	68 n:10ii
AND Rwn, [Rwi+]	68 n:11ii
CMP Rwn, Rwm	40 nm
CMP reg, #data16	46 RR ## ##
CMP reg, mem	42 RR MM MM
CMP Rwn, [Rwi]	48 n:10ii
CMP Rwn, [Rwi+]	48 n:11ii
JMPR cc, rel	cD rr
JMPS seg, caddr	FA ss MM MM
MOV reg, #data16	E6 RR ## ##
MOV Rwn, Rwm	F0 nm
MOV reg, mem	F2 RR MM MM
MOV mem, reg	F6 RR MM MM
MOV [Rwn], Rwm	B8 nm
MOV Rwn, [Rwm]	A8 nm
MOV Rwn, [Rwm+]	98 nm
SUB Rwn, Rwm	20 nm
SUB reg, #data16	26 RR ## ##
SUB reg, mem	22 RR MM MM
SUB mem, reg	24 RR MM MM
SUB Rwn, [Rwi]	28 n:10ii
SUB Rwn, [Rwi+]	28 n:11ii

Instruções ROM do uProcessador7 (Crivo de Eratóstenes)

Seg (ss)	Mnemonic	Format	Assembly
0	MOV reg, #data16	E6 RR ## ##	MOV R4, #0000h
1	MOV reg, #data16	E6 RR ## ##	MOV R0, #0020h
2	MOV reg, #data16	E6 RR ## ##	MOV R1, #0001h
3	MOV Rwn, Rwm	F0 nm	MOV R2, R1
4	JMPS seg, caddr	FA ss MM MM	JMPS #07h, #0000h
5	MOV [Rwm], Rwn	B8 nm	MOV [R2], R4
6	JMPS seg, caddr	FA ss MM MM	JMPS #0Dh, #0000h
7	MOV [Rwm], Rwn	B8 nm	MOV [R1], R1
8	ADD reg, #data16	06 nm	ADD R1, #0001h
9	CMP Rwn, Rwm	40 nm	CMP R1, R0
10	JMPR cc, rel	cD rr	JMPR #Fh, #03h
11	CMP reg, #data16	46 RR ## ##	CMP R2, #0002h
12	JMPR cc, rel	cD rr	JMPR #8h, #07h
13	MOV reg, #data16	E6 RR ## ##	MOV R2, #0002h
14	JMPS seg, caddr	FA ss MM MM	JMPS #10h, #0000h
15	JMPS seg, caddr	FA ss MM MM	JMPS #20h, #0000h
16	MOV Rwn, Rwm	F0 nm	MOV R1, R2
17	CMP Rwn, Rwm	40 nm	CMP R1, R0
18	JMPR cc, rel	cD rr	JMPR #2h, #03h
19	JMPS seg, caddr	FA ss MM MM	JMPS #1B, #0000h
20	MOV Rwn, Rwm	F0 nm	MOV R2, R1
21	ADD reg, #data16	06 nm	ADD R2, #0001h
22	CMP Rwn, Rwm	40 nm	CMP R2, R4
23	JMPR cc, rel	cD rr	JMPR #3h, #07h
24	CMP Rwn, Rwm	40 nm	CMP R2, R0
25	JMPR cc, rel	cD rr	JMPR #3h, #04h
26	JMPS seg, caddr	FA ss MM MM	JMPS #20h, #0000h
27	ADD Rwn, Rwm	01 nm	ADD R2, R1
28	CMP Rwn, Rwm	40 nm	CMP R2, R0
29	JMPR cc, rel	cD rr	JMPR #Eh, #09h

30	MOV [Rwm], Rwn	B8 nm	MOV [R2], R4
31	JMPS seg, caddr	FA ss MM MM	JMPS #1Bh, #0000h
32	MOV Rwn, Rwm	F0 nm	MOV R1, R0
33	MOV Rwn, [Rwm]	A8 nm	MOV R3, [R1]
34	ADD reg, #data16	06 nm	ADD R1, #0001h
35	CMP Rwn, Rwm	40 nm	CMP R1, R0
36	JMPR cc, rel	cD rr	JMPR #Fh, #03h
37	JMPS seg, caddr	FA ss MM MM	JMPS #25h, #0000h

Seg (ss)	Código de Máquina (Binário)	Código de Máquina (Hexadecimal)
0	1110 0110 0100 0000 0000 0000 0000 0000	E6400000
1	1110 0110 0000 0000 0000 0000 0020 0000	E6000020
2	1110 0110 0001 0000 0000 0000 0000 0001	E6100001
3	1111 0000 0010 0001 0000 0000 0000 0000	F0210000
4	1111 1010 0000 0111 0000 0000 0000 0000	FA070000
5	1011 1000 0010 0100 0000 0000 0000 0000	B8240000
6	1111 1010 0000 1101 0000 0000 0000 0000	FA0D0000
7	1011 1000 0001 0001 0000 0000 0000 0000	B8110000
8	0000 0110 0001 0000 0000 0000 0000 0001	06100001
9	0100 0000 0001 0000 0000 0000 0000 0000	40100000
10	1111 1101 0000 0011 0000 0000 0000 0000	FD030000
11	0100 0110 0010 0000 0000 0000 0000 0010	46200002
12	1000 1101 0000 0111 0000 0000 0000 0000	8D070000
13	1110 0110 0010 0000 0000 0000 0000 0010	E6200002
14	1111 1010 0001 0000 0000 0000 0000 0000	FA100000
15	1111 1010 0010 0000 0000 0000 0000 0000	FA200000
16	1111 0000 0001 0010 0000 0000 0000 0000	F0120000

17	0100 0000 0001 0000 0000 0000 0000 0000	40100000
18	0010 1101 0000 0011 0000 0000 0000 0000	2D030000
19	1111 1010 0001 1011 0000 0000 0000 0000	FA1B0000
20	1111 0000 0010 0001 0000 0000 0000 0000	F0210000
21	0000 0110 0010 0000 0000 0000 0000 0001	06200001
22	0100 0000 0010 0100 0000 0000 0000 0000	40240000
23	0011 1101 0000 0111 0000 0000 0000 0000	3D070000
24	0100 0000 0010 0000 0000 0000 0000 0000	40200000
25	0011 1101 0000 0100 0000 0000 0000 0000	3D040000
26	1111 1010 0010 0000 0000 0000 0000 0000	FA200000
27	0000 0001 0010 0001 0000 0000 0000 0000	01210000
28	0100 0000 0010 0000 0000 0000 0000 0000	40200000
29	1110 1101 0000 0111 0000 0000 0000 0000	ED090000
30	1011 1000 0010 0100 0000 0000 0000 0000	B8240000
31	1111 1010 0010 1011 0000 0000 0000 0000	FA1B0000
32	1111 0000 0001 0000 0000 0000 0000 0000	F0100000
33	1010 1000 0011 0001 0000 0000 0000 0000	A8310000
34	0000 0110 0001 0000 0000 0000 0000 0001	06100001
35	0100 0000 0001 0000 0000 0000 0000 0000	40100000
36	1111 1101 0000 0011 0000 0000 0000 0000	FD030000
37	1111 1010 0010 0101 0000 0000 0000 0000	FA250000

Observações

1º Byte corresponde a Op
2º Byte corresponde a n (registrador para SrcA e WD3) e m (SrcB)
3º e 4º Bytes ## ## para SignImm
3º e 4º Bytes MM MM para Address da RAM
2º Byte ss para Address da ROM
1º Byte c corresponde a condição:
EQ (c=2), NE (c=3), ULT (c=8), ULE (c=F), UGE (c=9), UGT (c=E)
3º e 4º Bytes rr correspondem a valor a ser convertido a complemento de dois
Rw é registrador word de general purpose
reg, neste caso, é utilizado como registrador word (poderia ser reg byte, não utilizado neste código)
mem corresponde ao endereço de memória (MM MM, mas utiliza-se apenas dos MM menos significativos)
rel corresponde ao valor que será convertido em complemento de dois
:... os 2 primeiros bits indicam presença ou não de offset (11 para sim e 10 para não) enquanto os dois últimos, registradores de 0 a 3
[Rwi] representa valor indireto de registrador para endereçamento da RAM e [Rwi+] valor de endereçamento com offset

Referências

Programming Manual

https://www.st.com/content/ccc/resource/technical/document/programming_manual/20/09/13/1c/cf/32/4b/ce/CD00004609.pdf/files/CD00004609.pdf/jcr:content/translations/en.CD00004609.pdf