Progress report about RISC architecture development

Instruction Set of Designed RISC.

Designed RISC contains 6 types of command:

1) Unconditional load to register
2) Unconditional store from register to memory
3) Unconditional jump
4) Conditional jump
5) ALU commands
6) Conditional load to register from registers

Format of Unconditional load to register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Type		Offset	INCR SRC2			x	x	x	x	x	x	x	Reg SRC2					Reg SRC1					Reg DST				

[27:26] type of LDR operations
0 - Rdst= Rsrc1
1 - Rdst[31:16] = code[20:5] LDRH Rx,0x1234
2 - Rdst[15:0] = code[20:5] LDRL Rx,0x5678
3 - Rdst=[Rsrc1+offset]
[25]
0 - offset = code[24:10](signed)
1 - offset = Rsrc2(signed)
[24:22]
000 Rsrc2=Rsrc2
001 Rsrc2=Rsrc2+1
010 Rsrc2=Rsrc2+2
011 Rsrc2=Rsrc2+4
100 Rsrc2=Rsrc2
101 Rsrc2=Rsrc2-1
110 Rsrc2=Rsrc2-2
111 Rsrc2=Rsrc2-4
[14:10] src2
[9:5]src1
[4:0]dst

Format of Unconditional store from register to memory

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	x	x	Offset	INCR SRC2			x	x	x	x	x	x	x	Reg SRC2					Reg SRC1					Reg DST				

[27:26] type of STORE operation
XX [Rsrc1+offset]=Rdst
[25]
0 - offset = code[24:10](signed)
1 - offset = Rsrc2(signed)
[24:23]
000 Rsrc2=Rsrc2
001 Rsrc2=Rsrc2+1
010 Rsrc2=Rsrc2+2
011 Rsrc2=Rsrc2+4
100 Rsrc2=Rsrc2
101 Rsrc2=Rsrc2-1
110 Rsrc2=Rsrc2-2
111 Rsrc2=Rsrc2-4
[14:10] src2
[9:5]src1
[4:0]dst

Format of Unconditional jumps

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	Type		Offset	INCR SRC2			x	x	x	x	x	x	x	Reg SRC2					Reg SRC1					Reg DST				

[27:26] type of jumps
0 – jmp pc[25:0]=code[25:0]
1 – jmp pc=pc+offset(relaitive jump) jmpr R2(jmpr R2+4)
2 - call pc=pc+offset, Rdst=pc callr R0,0xXXXXXXX,R1+4 or callr R0,R1+4
3 - ret pc=Rdst ret Rx or ret Rx,Ry+-0,1,2,4
[25]
0 - offset = code[24:10](signed)
1 - offset = Rsrc2(signed)
[24:23]
000 Rsrc2=Rsrc2
001 Rsrc2=Rsrc2+1
010 Rsrc2=Rsrc2+2
011 Rsrc2=Rsrc2+4
100 Rsrc2=Rsrc2
101 Rsrc2=Rsrc2-1
110 Rsrc2=Rsrc2-2
111 Rsrc2=Rsrc2-4
[14:10] src2
[9:5]src1
[4:0]dst

Format of Conditional jumps

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	Type		Offset	INCR SRC2			x	x	x	x	x	x	x	Reg SRC2					Reg SRC1					Reg DST				

[27:26] type of jumps
0 - jmpz pc=pc+offset
1 - jmpnz pc=pc+offset
2 - jmpc pc=pc+offset
3 - jmpnc pc=pc+offset
[25]
0 - offset = code[24:10](signed)
1 - offset = Rsrc2(signed)
[24:22]
000 Rsrc2=Rsrc2
001 Rsrc2=Rsrc2+1
010 Rsrc2=Rsrc2+2
011 Rsrc2=Rsrc2+4
100 Rsrc2=Rsrc2
101 Rsrc2=Rsrc2-1
110 Rsrc2=Rsrc2-2
111 Rsrc2=Rsrc2-4
[14:10] src2
[9:5]src1
[4:0]dst

Format of ALU operations

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	Type			INCR SRC2			x	x	x	x	x	x	x	Reg SRC2					Reg SRC1					Reg DST				

[27:25] type of op
0- AND
1- OR
2- XOR
3- ADD
4- MUL
5-shift Rsrc1 left by Rscr2 ...0 MSB->C flag
6-shift Rsrc1 right by Rscr2 ...0 LSB ->C flag
7-CMP compare,
[24:22]
000 Rsrc2=Rsrc2
001 Rsrc2=Rsrc2+1
010 Rsrc2=Rsrc2+2
011 Rsrc2=Rsrc2+4
100 Rsrc2=Rsrc2
101 Rsrc2=Rsrc2-1
110 Rsrc2=Rsrc2-2
111 Rsrc2=Rsrc2-4
[14:10] src2
[9:5]src1
[4:0]dst

Format of Conditional load to register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Type		Offset	INCR SRC2			x	x	x	x	x	x	x	Reg SRC2					Reg SRC1					Reg DST				

[27:26] type of LDRF op
0-LDRZ Rdst=Rsrc1 if z=1, otherwise Rdst=Rsrc2
1-LDRNZ Rdst=Rsrc1 if z=0, otherwise Rdst=Rsrc2
2-LDRC Rdst=Rsrc1 if c=1, otherwise Rdst=Rsrc2
3-LDRNC Rdst=Rsrc1 if c=0, otherwise Rdst=Rsrc2
[24:22]
000 Rsrc2=Rsrc2
001 Rsrc2=Rsrc2+1
010 Rsrc2=Rsrc2+2
011 Rsrc2=Rsrc2+4
100 Rsrc2=Rsrc2
101 Rsrc2=Rsrc2-1
110 Rsrc2=Rsrc2-2
111 Rsrc2=Rsrc2-4
[14:10] src2
[9:5]src1
[4:0]dst

Sorting algorithm performance.

Chosen Shell sort algorithm:

```
ShellSort(int * a)
{
	int inc,i,j,temp;

	for(inc = 5;inc>0;inc=inc/2)
	{
		for (i = inc; i < 10; i+=1)
		{
			temp = a[i];
			for (j = i; (j >= inc) && (a[j-inc] > temp); j =j-inc)
			{
				a[j] = a[j-inc];
			}
		a[j] = temp;
	}
}
```

Modified shell sort

```
	//new algorithm
	for(inc = 20;inc>0;inc=inc/2)//20, 8 and 4 (5,2,1)
	{
		for (i = inc; i < 40; i+=4)//0..4..8..c and etc
		{
			temp = a[i];
			for (j = i; (j >= inc) && (a[j-inc] > temp); j =j-inc)
			{
				a[j] = a[j-inc];
			}
		a[j] = temp;
		}
	}
```

Bubble sort shown best performance bellow you can see algorithm in psevdocode. Psevdocode is close to assembler language of designed RISC. Cycles of bubble sort are 455 for any type of sorted data.

```
//bubble algorithm
	R0=mem[00],R1=mem[04],R2=mem[08],R3=mem[c],R4=mem[10]
	R5=mem[14],R6=mem[18],R7=mem[1c],R8=mem[20],R9=mem[24]
	R10=R0;
	R11=R0;//minimum
	if R1<R11 then
		R11=R1
	if R2<R11 then
		R11=R2
	if R3<R11 then
		R11=R3
	if R4<R11 then
		R11=R4
	if R5<R11 then
		R11=R5
	if R6<R11 then
		R11=R6
	if R7<R11 then
		R11=R7
	if R8<R11 then
		R11=R8
	if R9<R11 then
		R11=R9

	mem[0]=R11;

	for (i = 36; i > 0; i-=4)//36,32,28,24,20,16,12,8,4
	{
		if R0>R10 then
			R10=R0
		if R1>R10 then
```

```
                R10=R1
        if R2>R10 then
                R10=R2
        if R3>R10 then
                R10=R3
        if R4>R10 then
                R10=R4
        if R5>R10 then
                R10=R5
        if R6>R10 then
                R10=R6
        if R7>R10 then
                R10=R7
        if R8>R10 then
                R10=R8
        if R9>R10 then
                R10=R9

        mem[i]=R10;

        //substitute maximum by minimum
        if R10=R0 then
                R0=R11
        if R10=R1 then
                R1=R11
        if R10=R2 then
                R2=R11
        if R10=R3 then
                R3=R11
        if R10=R4 then
                R4=R11
        if R10=R5 then
                R5=R11
        if R10=R6 then
                R6=R11
        if R10=R7 then
                R7=R11
        if R10=R8 then
                R8=R11
        if R10=R9 then
                R9=R11

}
```