Zaawansowane typy instrukcji

Sekcja 7.

Instrukcje rownolegle 1/3

1. *compute*, | DM(Ia, Mb) = dreg1
dreg1 = DM(Ia, Mb) | , | PM(Ic, Md) = dreg2
dreg2 = PM(Ic, Md) | ;

2. *IF condition* *compute* ;

3a. *IF condition* *compute*, | DM(Ia, Mb)
PM(Ic, Md) | = ureg ;

3b. *IF condition* *compute*, | DM(Mb, Ia)
PM(Md, Ic) | = ureg ;

3c. *IF condition* *compute*, ureg = | DM(Ia, Mb)
PM(Ic, Md) | ;

3d. *IF condition* *compute*, ureg = | DM(Mb, Ia)
PM(Md, Ic) | ;

Instrukcje rownolegle 2/3

4a. *IF condition* *compute*, | DM(Ia, <data6>) PM(Ic, <data6>) | = dreg ;

4b. *IF condition* *compute*, | DM(<data6>, Ia) PM(<data6>, Ic) | = dreg ;

4c. *IF condition* *compute*, dreg = | DM(Ia, <data6>) PM(Ic, <data6>) | ;

4d. *IF condition* *compute*, dreg = | DM(<data6>, Ia) PM(<data6>, Ic) | ;

5. *IF condition* *compute*, ureg1 = ureg2 ;

6a. *IF condition* *shiftimm,* | DM(Ia, Mb) PM(Ic, Md) | = dreg ;

6b. *IF condition* *shiftimm,* dreg = | DM(Ia, Mb) PM(Ic, Md) | ;

7. *IF condition* *compute,* MODIFY | (Ia, Mb) (Ic, Md) | ;

- **<data6> is a 6 bit field in the opcode; you can represent +/- 32 integers**

Instrukcje rownolegle 3/3

Parallel Program Flow and Computes (SHARC)

IF condition JUMP | (M8-M15,I8-I15) / (PC,<reladdr6>) | (| DB / LA / CI / DB,LA / DB,CI |) , | *compute* / *ELSE compute* | ;

IF condition CALL | (M8-M15,I8-I15) / (PC,<reladdr6>) | (DB) , | *compute* / *ELSE compute* | ;

IF condition JUMP | (M8-M15,I8-I15) / (PC,<reladdr6>) | ELSE , | *compute,* DM(M0-M7,I0-I7)=dreg / *compute,* dreg=DM(M0-M7,I0-I7) | ;

IF condition | RTS / RTI | (| DB / LR / DB,LR |) , | *compute* / *ELSE compute* | ;

(DB) Delayed Branch
(LR) Loop reentry

(LA) Loop abort (pop loop and PC stacks)
(CI) Clear Interrupt

Warunkowy rejestr Swap Hint

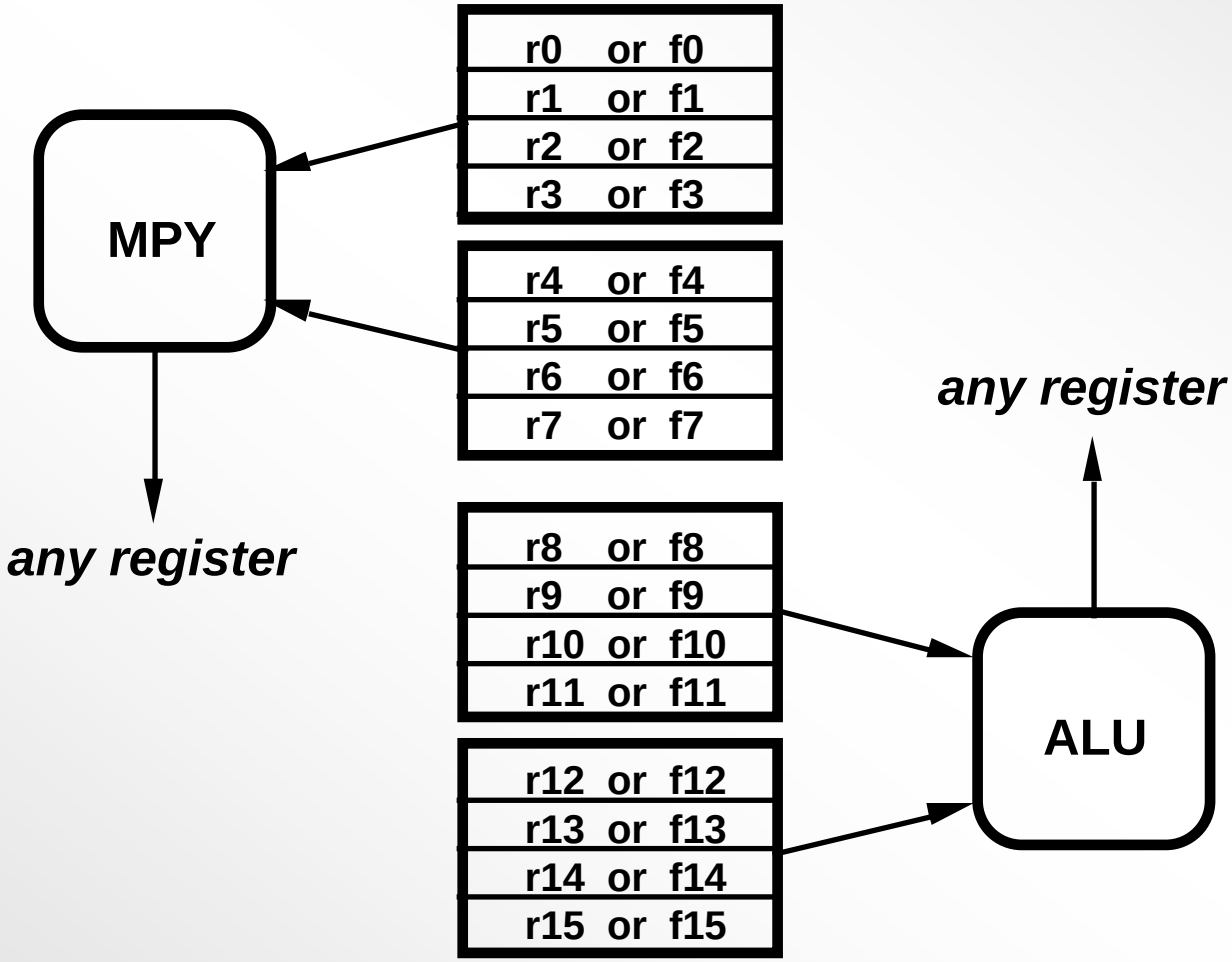
- Aby zamienic dwa rejestry nalezy uzyc instrukcji PASS w rejestrze transferu:

```
if le r0 = pass r1 , r1 = r0 ;
```

- Zamiast:

```
if le r2 = r0;
if le r0 = r1;
if le r1 = r2;
```

Rejestr danych dla obliczen wielofunkcyjnych



Obliczenia wielofunkcyjne

Fixed-Point Multiply/Accumulate and Add, Subtract, or Average

Rm =R3-0 * R7-4(SSFR) MRF=MRF + R3-0 * R7-4(SSF) Rm =MRF + R3-0 * R7-4(SSFR) MRF=MRF - R3-0 * R7-4(SSF) Rm =MRF - R3-0 * R7-4(SSFR)	,	Ra= R11-8 + R15-12 Ra= R11-8 - R15-12 Ra=(R11+8 + R15-12)/2	;

Floating-Point Multiplication and ALU Operation

Fm=F3-0 * F7-4,	Fa=F11-8 + F15-12 Fa=F11-8 - F15-12 Fa=FLOAT R11-8 by R15-12 Fa=FIX R11-8 by R15-12 Fa=(F11-8+F15-12)/2 Fa=ABS F11-8 Fa=MAX (F11-8, F15-12) Fa=MIN (F11-8,F15-12)	;

Multiplication and Dual Add/Subtract

Rm=R3-0 * R7-4(SSFR),	Ra=R11-8 + R15-12,	Rs=R11-8 - R15-12 ;
Fm=F3-0 * F7-4,	Fa=F11-8 + F15-12,	Fs=F11-8 - F15-12 ;

Obliczenia wielofunkcyjne i rownolegle:

Przyklad Radix-2 FFT Butterfly

```
        LCNTR=r15, Do end_bfly until LCE;
           f8=f1*f6,                 f14=f11-f14,  dm(i2,m0)=f10,  f9=pm(i11,m8);
           f11=f1*f7,  f3=f9+f14,    f9=f9-f14,    dm(i2,m0)=f13,  f7=pm(i8,m8);
           f14=f0*f6,  f12=f8+f12,                 f8=dm(i0,m0),   pm(i10,m10)=f9;
end_bfly:  f12=f0*f7,  f13=f8+f12,  f10=f8-f12,   f6=dm(i0,m0),   pm(i10,m10)=f3;
```

Funkcje obustronne i podzialu

- Syntax: `Fn = RECIPS Fx`
- Creates an 8 bit accurate floating-point seed for 1/Fx from an internal ROM lookup table
- An Iterative Convergence Algorithm can obtain
  - Single-Precision Accuracy (24 bit mantissa) in 6 cycles
  - Extended-Precision Accuracy (32 bit mantissa) in 8 cycles
- Divide can be obtained by one additional multiply which can be done with no extra cycles in parallel with an ALU op.
- `RTS` can even be done in parallel, allowing a function return

Single Precision Divide Subroutine

Q = N / D

- **f0 = Numerator, f12 = Denominator, F11=2.0, f0 = Quotient**

```
f0=recips f12, f7=f0; /*Get 8 bit seed R0=1/D*/
f12=f0*f12;           /*D'=D*R0 */
f7=f0*f7, f0=f11-f12; /*f0=r1=2-D',f7=N*r0*/
rts (db), f12=f0*f12; /*f12=D'=D'*r2*/
f7=f0*f7, f0=f11-f12; /*f7=N*R0*R1*r2,f0=2-D'*/
f0=f0*f7;             /*f0=n*R0*R1*R2*R3*/
```

Funkcja 1/Square Root

- Syntax: `Fn = RSQRTS Fx`
- 4 bitowa dokladnosc zmienno-przecinkowa ustawiona na 1/sqrt(Fx) z wewnetrznej ROM lookup table
- algorytmem iteracji Newtona-Raphsona mozna otrzymac
  - dokladnosc pojedynczej precyzji (24 bitowa mantysa) w 9 cyklach
  - dokladnisc rozszerzonej precyzji (32 bitowa mantysa) w13 cyklach
- Square Root moze byc uzyskany przy jednokrotnym mnozeniu
- `RTS` moze byc wykonane rownolegle, co pozwala na zwrot wartosci

Cwiczenie programowanie #2

LAB 12

6 cykli na petle (lacznie 122 cykle)

```
lcntr = LENGTH, do loopend until lce;
            f0 = dm(i0,m1);          /* read A*/
            f1 = pm(i8,m9);          /*read B */
            f3 = f0 * f1;            /* calculate A * B*/
            f4 = dm(i1,m1);          /*read C */
            f3 = f3 + f4;            /* add C to product */
loopend:   dm(i2,m1) = f3;           /* store result */
```

4 cykle na petle (lacznie 82 cykle)

```
lcntr = LENGTH, do loopend until lce;
      f0 = dm(i0,m1),  f1 = pm(i8,m9);     /* read A, read B */
      f3 = f0 * f1,  f4 = dm(i1,m1);       /* calculate A * B, read C */
      f3 = f3 + f4;                        /* add C to product */
loopend:   dm(i2,m1) = f3;                 /* store result */
```

3 cykle na petle (lacznie 64 cykli)

```
f0=dm(i0,m1), f1=pm(i8,m9);          /* read A, read B */

lcntr=LENGTH-1, do loopend until lce;
      f3=f0*f1,  f4=dm(i1,m1);       /* calc A * B, read C */
      f3=f3+f4,  f0=dm(i0,m1),  f1=pm(i8,m9);    /* add C, read A, B */
loopend: dm(i2,m1)=f3;               /* store result */

f3=f0*f1, f4=dm(i1,m1);              /* calc A * B, read C */
f3=f3+f4;                            /* add C to product */
dm(i2,m1)=f3;                        /* store result */
```

2 cykle na petle (lacznie 50 cykli)

```
f0=dm(i0,m1),  f1=pm(i8,m9);                    /* read A, read B */
f3=f0*f1,  f4=dm(i1,m1);                        /* calc A * B, read C */
f3=f3+f4,  f0=dm(i0,m1), f 1=pm(i8,m9);         /* add C, read A, B */

lcntr=LENGTH-2, do loopend until lce;
   f3=f0*f1,  f4=dm(i1,m1),  pm(i9,m9)=f3;      /* A*B,read C,store result */
loopend: f3=f3+f4,  f0=dm(i0,m1),  f1=pm(i8,m9);/* add C, read A, B */

f3=f0*f1,  f4=dm(i1,m1),  pm(i9,m9)=f3;         /* A*B,read C,store result */
f3=f3+f4;                                       /* add C to product */
pm(i9,m9)=f3;                                   /* store result */
```