1-1

fib.mc — microcode for Fibonacci (85 cycles)

```
        {MBR=0}
START:  mar := sp ; d := 1 + sp ; wr;        0|00|00|00|0|1|0|1|1| d|sp
          { D = nextaddr (SP+1) }
        f := 1 + 1; wr;
          { F=2 ; D = SP+1 ; M[SP] = Fib(0) }
        mbr := 1; mar := d; wr; b := 1;
          { F=2, D=SP+1, M[SP]=Fib(0), B= Fib(1) }   00010001101 1 b d '1' 0
        e := f + 1; wr;
          { F=2, E=3, D=SP+1, M[SP] = A = Fib(0); M[SP+1] = B = Fib(1) }
LOOP:   { Invariant: A=Fib(n) B=Fib(n+1) SP = SP0 + n ; }
        a := b + a;
          { A = Fib(n+2) B = Fib(n+1) }
        d := sp + f;
        mar := d; mbr := a; wr; if n then goto DONE;
          { in 1 cycle, M[SP+2] = Fib(n+2) }
        sp := e + sp ; wr;
          { M[SP-1] = Fib(n+2) }
        b := a + b;
          { A = Fib(n+2) B = Fib(n+3) }
        mar := sp ; mbr := b; wr; if n then goto DONE;
        wr; goto LOOP; sp := d;
DONE:   wr; rd;
```

VLIW - Very Long Instruction Word

```
9-2/ Micro Program := ____
                      | NL    mp
                      | LABEL  ":"  NL  mp
                      | Instruction  NL  mp
   Instruction := ____
                | Component  ";"  Instruction
  Component :=  "mar :="   BExpr              — set MAR,  set B
             |  "mbr :="   ShExpr             — set MBR, set SH, ALU, A, B, AMUX
             |  Reg  ":="  ShExpr             — set ENC, C, /
             |  "alu :="   AluExpr            — set ALU, A, B, AMUX
             |  "wr"                          — set WR
             |  "rd"                          — set RD
             |  "goto"  LABEL                 — set COND=11, ADDR=LABEL
             |  "if" Cond "then goto" LABEL   — set COND = n/z , ADDR = LABEL
 Cond := N (01) | Z (10)
 ShExpr :=  AluExpr                           — set SH=00, set ALU, A, B, AMUX
        |  "lshift(" AluExpr ")"              —     SH=10           ''
        |  "rshift(" AluExpr ")"              —     SH=01           ''
 AluExpr :=  Axpr "+" BExpr                   — ALU=00 , set A, AMUX, B
         |   AExpr                            — ALU=10 , set A, AMUX
         |   "inv(" Aexpr ")"                 — ALU=11 , set A, Amx
         |   "band(" Aexpre ")" Bexpr ")"     — ALU=01 , set A, Amx, B
 AExpr  :=  Register                          — A = reg   AMUX=0
        |  "mbr"                              — AMUX = 1
 BExpr  :=  Register                          — B=reg
```

```
illegal :   mar = 1 ;  ac := ac + ac ; ] = MAR=1, ENC=1, AMUX=0, A=AC, B=AC, C=AC
                                          B=1
```