Y86 Instruction Set

Each register has 4-bit ID as follows. Register ID 8 indicates “no register”.

<table>
<thead>
<tr>
<th>Register</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td>0</td>
</tr>
<tr>
<td>%ecx</td>
<td>1</td>
</tr>
<tr>
<td>%edx</td>
<td>2</td>
</tr>
<tr>
<td>%ebx</td>
<td>3</td>
</tr>
<tr>
<td>%esi</td>
<td>6</td>
</tr>
<tr>
<td>%edi</td>
<td>7</td>
</tr>
<tr>
<td>%esp</td>
<td>4</td>
</tr>
<tr>
<td>%ebp</td>
<td>5</td>
</tr>
</tbody>
</table>

irmovl V, rB

<table>
<thead>
<tr>
<th>Fetch</th>
<th>iCode:ifun</th>
<th>iCode:ifun ← M_i[PC]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rA, rB</td>
<td>rA:rB ← M_i[PC+1]</td>
</tr>
<tr>
<td></td>
<td>valC</td>
<td>valC ← M_i[PC+2]</td>
</tr>
<tr>
<td></td>
<td>valP</td>
<td>valP ← PC+6</td>
</tr>
</tbody>
</table>

| Decode         | valA, srcA | Compute next PC |
|                | valB, srcB |

| Execute        | valE       | Perform ALU operation |
|                | valE ← 0 + valC |

| Memory         | valM       | Write back ALU result |

| Write back     | dstE       | Update PC |
|                | dstM       |

<p>| PC update      | PC         |                       |
|                | PC ← valP  |</p>
<table>
<thead>
<tr>
<th>Stage</th>
<th>Operation</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td><code>icode:ifun ← M_i[PC]</code></td>
<td>Read instruction byte</td>
</tr>
<tr>
<td></td>
<td><code>valC ← M_i[PC+1]</code></td>
<td>Read destination address</td>
</tr>
<tr>
<td></td>
<td><code>valP ← PC+5</code></td>
<td>Fall through address</td>
</tr>
<tr>
<td>Decode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td><code>Bch ← Cond(CC,ifun)</code></td>
<td>Take branch?</td>
</tr>
<tr>
<td>Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write back</td>
<td><code>PC ← Bch ? valC : valP</code></td>
<td>Update PC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stage</th>
<th>Operation</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td><code>icode:ifun ← M_i[PC]</code></td>
<td>Read instruction byte</td>
</tr>
<tr>
<td></td>
<td><code>rA:rB ← M_i[PC+1]</code></td>
<td>Read register byte</td>
</tr>
<tr>
<td></td>
<td><code>valP ← PC+2</code></td>
<td>Compute next PC</td>
</tr>
<tr>
<td>Decode</td>
<td><code>valA ← R[rA]</code></td>
<td>Read operand A</td>
</tr>
<tr>
<td></td>
<td><code>valB ← R[rB]</code></td>
<td>Read operand B</td>
</tr>
<tr>
<td>Execute</td>
<td><code>valE ← valB OP valA</code></td>
<td>Perform ALU operation</td>
</tr>
<tr>
<td></td>
<td><code>Set CC</code></td>
<td>Set condition code register</td>
</tr>
<tr>
<td>Memory</td>
<td></td>
<td>Write back result</td>
</tr>
<tr>
<td>Write back</td>
<td><code>R[rB] ← valE</code></td>
<td>Update PC</td>
</tr>
<tr>
<td>PC update</td>
<td><code>PC ← valP</code></td>
<td></td>
</tr>
</tbody>
</table>
Execute stage computations

int sig valE 'valE'

Value computed by ALU
bool sig Bel 'bcond'
Branch test

Memory stage computations

int sig valM 'valM'

Value read from memory

Fetch stage computations

int sig valA 'valA'
Value from register A port
int sig valB 'valB'
Value from register B port

Fetch stage inputs

int sig pc 'pc'
Program counter
int sig icode 'icode'
Instruction control code
int sig ifun 'ifun'
Instruction function
int sig rA 'rA'
R field from instruction
int sig rb 'rb'
B field from instruction
int sig vaC 'vaC'
Constant value
int sig vaP 'vaP'
Address of following instruction

Fetch stage computations

int sig icode 'icode'
Instruction control code
int sig ifun 'ifun'
Instruction function
int sig rA 'rA'
R field from instruction
int sig rb 'rb'
B field from instruction
int sig vaC 'vaC'
Constant value
int sig vaP 'vaP'
Address of following instruction

#include "sim.h"

int main(int argc, char *argv[])
{
    int genc =
    int pc =
    int dstE =
    int dstM =
    # Does fetched instruction require a regid byte?
    bool needvalC =
    
    # Does fetched instruction require a constant word?
    bool instr_valid =
    
    # What register should be used as the A source?
    int srcA =
    
    # What register should be used as the B source?
    int srcB =
    
    # What register should be used as the E destination?
    int dstE =
    
    # What register should be used as the M destination?
    int dstM =

    # Symbolic representation of Y86 Instruction Codes
    #
    int sig INOP 'INOP'
    int sig IHALT 'IHALT'
    int sig IRRMOVL 'IRRMOVL'
    int sig IMRMMOVLO 'IMRMMOVLO'
    int sig IOPL 'IOPL'
    int sig LJXX 'LJXX'
    int sig ICALL 'ICALL'
    int sig IRET 'IRET'
    int sig IPUSHL 'IPUSHL'
    int sig IPOPL 'IPOPL'
    int sig ILEAVE 'ILEAVE'
    
    # Instruction code for iaddl instruction
    int sig IADDL 'IADDL'
    
    # Instruction code for leave instruction
    int sig ILEAVE 'ILEAVE'
    
    # Symbolic representation of Y86 Registers referenced explicitly
    #
    int sig REG 'REG'
    int sig RESP 'RESP'
    int sig REBP 'REBP'
    int sig RNONE 'RNONE'
    int sig NONE 'NONE'
    int sig RDST 'RDST'
    int sig RSOURCE 'RSOURCE'
    
    # ALU Functions referenced explicitly
    #
    int sig ALUADD 'ALUADD'
    int sig ALUADD 'ALUADD'
    
    # Fetch stage inputs
    #
    int sig pc 'pc'
    int sig icode 'icode'
    int sig ifun 'ifun'
    int sig rA 'rA'
    int sig rb 'rb'
    int sig vaC 'vaC'
    int sig vaP 'vaP'
    
    # Fetch stage outputs
    #
    int sig dstE 'dstE'
    int sig dstM 'dstM'
    int sig srcA 'srcA'
    int sig srcB 'srcB'
    
    # Fetch stage computations
    #
    int sig icode 'icode'
    int sig ifun 'ifun'
    int sig rA 'rA'
    int sig rb 'rb'
    int sig vaC 'vaC'
    int sig vaP 'vaP'
    
    # Execute stage computations
    #
    int sig valA 'valA'
    int sig valB 'valB'
    int sig bel 'bcond'
    
    # Memory stage computations
    #
    int sig valM 'valM'
    
    # Control Signal Definitions
    #
    # Fetch Stage
    #
    # What registers should be used as the A source?
    # What registers should be used as the B source?
    # What registers should be used as the E destination?
    # What registers should be used as the M destination?
```c
bool mem_read = icode in { BRMMOV, IPOPL, IRET, ILEAVE };  // Set read control signal

bool mem_write = icode in { BRMMOV, IPUSHL, ICALL };  // Select memory address
int mem_addr = [  
    icode in { BRMMOV, IPUSHL, ICALL, BRMMOV } : valE;  
    icode in { IPUSHL, IRET } : valA;  
    icode in { ILAVE } : valA;  
    # Other instructions don't need address
];

bool mem_read = icode in { BRMMOV, IPOPL, IRET, ILEAVE };  // Set write control signal
bool mem_write = icode in { BRMMOV, IPUSHL, ICALL };  // Select memory input data
int mem_data = [  
    # Value from register  
    icode in { BRMMOV, IPUSHL } : valA;  
    # Return PC  
    icode in { ICALL } : valP;  
    # Default: Don't write anything
];

bool mem_read = icode in { BRMMOV, IPOPL, IRET, ILEAVE };  // Program Counter Update
int new_pc = [  
    # Call. Use instruction constant  
    icode in { ICALL } : valC;  
    # Taken branch. Use instruction constant  
    icode in { IJXX & & Bch } : valC;  
    # Completion of RET instruction. Use value from stack  
    icode in { IRET } : valM;  
    # Default: Use incremented PC  
    1 : valP;  
];
```
# Does fetched instruction require a regid byte?
bool need_regid = fcode in {BRMOVCL, IOPL, IPUSHL, IPOLP, BRMOVCL, BRMOVCL, BRMOVCL, BRMOVCL, HADDDL};

# Does fetched instruction require a constant word?
bool new_destE = fcode in {BRMOVCL, BRMOVCL, BRMOVCL, IJX, ICALL, HADDDL};

bool instr_valid = fcode in {INOP, HALT, BRMOVCL, BRMOVCL, BRMOVCL, BRMOVCL, BRMOVCL, IOPL, IJX, ICALL, RET, IPUSHL, IPOLP, HADDDL, ILEAVE};

# Predict next value of PC
int new_P_predPC = fcode in {IJXX, ICALL}: fvalC; 1 : fvalP;

### Decode Stage ###

#### What register should be used as the A source? ####
int new_destA = |
Dcode in {BRMOVCL, BRMOVCL, IOPL, IPUSHL}: DvalA;
Dcode in {IOPL, RET}: RESP;
Dcode in {ILEAVE}: RESP;
1 : RNONE; # Don’t need register

#### What register should be used as the B source? ####
int new_destB = |
Dcode in {IOPL, BRMOVCL, BRMOVCL, HADDDL}: DvalB;
Dcode in {IPUSHL, ILEAVE, IRET} : RESP;
Dcode in {ILEAVE}: RESP;
1 : RNONE; # Don’t need register

#### What register should be used as the E destination? ####
int new_destE = |
Dcode in {BRMOVCL, BRMOVCL, IOPL, HADDDL}: DvalE;
Dcode in {IPUSHL, IOPL, ICALL, IRET, ILEAVE}: RESP;
1 : RNONE; # Don’t need register

#### What register should be used as the M destination? ####
int new_destM = |
Dcode in {BRMOVCL, IOPL}: DvalM;
Dcode in {ILEAVE}: RESP;
1 : RNONE; # Don’t need register

#### What should be the A value? ####
# Forward into decode stage for valA
int new_srcA = |
Dcode in {ICALL, IJX}: DvalAP; # Use incremented PC
d_srcC = E_dstE : e_valE; # Forward valE from execute
d_srcB = M_intM : m_valM; # Forward valM from memory
d_srcA = M_intE : m_valE; # Forward valE from memory
d_srcC = W_intE : W_valE; # Forward valE from write back
d_srcA = W_dstE : W_valE; # Forward valE from write back
1 : DvalB; # Use value read from register file

int new_valB = |
d_srcC = E_dstE : e_valE; # Forward valE from execute
d_srcB = M_intM : m_valM; # Forward valM from memory
d_srcB = M_intE : m_valE; # Forward valE from memory
d_srcC = W_intE : W_valE; # Forward valE from write back
d_srcA = W_dstE : W_valE; # Forward valE from write back
1 : DvalB; # Use value read from register file

int new_valA = |
d_srcC = E_dstE : e_valE; # Forward valE from execute
d_srcB = M_intM : m_valM; # Forward valM from memory

#### Memory Stage ####

#### Should the condition codes be updated? ####
bool set_CC = E_code in {IOPL, HADDDL};

#### Memory address ####
int mem_addr = |
M_code in {BRMOVCL, IPUSHL, ICALL, BRMOVCL}: M_valE;
M_code in {IOPL, IRET, ILEAVE}: M_valA;
# Other instructions don’t need address

#### Set read control signal ####
bool mem_read = M_code in {BRMOVCL, IPUSHL, ICALL};

#### Set write control signal ####
bool mem_write = M_code in {BRMOVCL, IPUSHL, ICALL};

#### Pipeline Register Control ####
# Should I stall or inject a bubble into Pipeline Register F?
# At most one of these can be true.
bool F_stall = 0;
bool F_bubble = 0;

### Execute Stage ###

#### Select input A to ALU ####
int aluA = |
E_code in {BRMOVCL, IOPL, ICALL, BRMOVCL, HADDDL}: E_valC;
E_code in {ICALL, IPUSHL, IPOLP, HADDDL, ILEAVE}: -4;
# Other instructions don’t need ALU

#### Select input B to ALU ####
int aluB = |
E_code in {BRMOVCL, BRMOVCL, IOPL, ICALL, IPUSHL, IRET, IPOLP, HADDDL, ILEAVE}: E_valB;
E_code in {IPUSHL, IRET, ILEAVE}: 0;
# Other instructions don’t need ALU

#### Set the ALU function ####
int alufun = |
E_code == IOPL: E_valB;
1: ALUADD;

#### Should the condition codes be updated? ####
bool set_CC = E_code in {IOPL, HADDDL};

#### Set memory address ####
int mem_addr = |
M_code in {BRMOVCL, IPUSHL, ICALL, BRMOVCL}: M_valE;
M_code in {IOPL, IRET, ILEAVE}: M_valA;
# Other instructions don’t need address

#### Set read control signal ####
bool mem_read = M_code in {BRMOVCL, IPUSHL, ICALL};

#### Set write control signal ####
bool mem_write = M_code in {BRMOVCL, IPUSHL, ICALL};
E\text{code} \in \{\text{IMMOV, IPOPL, ILEAVE}\} \land
E\text{dstM} \in \{d_{srcA}, d_{srcB}\} \land
\# \text{Stalling at fetch while ret passes through pipeline}
IRET \in \{D\text{code}, E\text{code}, M\text{code}\};
\# \text{ Should I stall or inject a bubble into Pipeline Register D?}
\# \text{At most one of these can be true.}
bool D\text{stall} =
\# \text{Conditions for a load/use hazard}
E\text{code} \in \{\text{IMMOV, IPOPL, ILEAVE}\} \land
E\text{dstM} \in \{d_{srcA}, d_{srcB}\} ;

bool D\text{bubble} =
\# \text{Mispredicted branch}
(E\text{code} == \text{IJXX} \land \neg Bch) ||
\# \text{Stalling at fetch while ret passes through pipeline}
\# \text{but not condition for a load/use hazard}
\neg(E\text{code} \in \{\text{IMMOV, IPOPL, ILEAVE}\} \land E\text{dstM} \in \{d_{srcA}, d_{srcB}\} ) \land

IRET \in \{D\text{code}, E\text{code}, M\text{code}\};
\# \text{Should I stall or inject a bubble into Pipeline Register E?}
\# \text{At most one of these can be true.}
bool E\text{stall} = 0;
bool E\text{bubble} =
\# \text{Mispredicted branch}
(E\text{code} == \text{IJXX} \land \neg Bch) ||
\# \text{Conditions for a load/use hazard}
E\text{code} \in \{\text{IMMOV, IPOPL, ILEAVE}\} \land
E\text{dstM} \in \{d_{srcA}, d_{srcB}\} ;

# \text{ Should I stall or inject a bubble into Pipeline Register M?}
# \text{At most one of these can be true.}
bool M\text{stall} = 0;
bool M\text{bubble} = 0;
#/* $end\ pipe-all-hcl */