ΗΥ340 : ΓΛΩΣΣΕΣ ΚΑΙ ΜΕΤΑΦΡΑΣΤΕΣ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ,
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ,
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

```
VAR i:Integer;
    FUNCTION(Symbol) replicate
        x = (function(x,y){return x+y;});
          class DelFunctor: public std::unary_function<
```

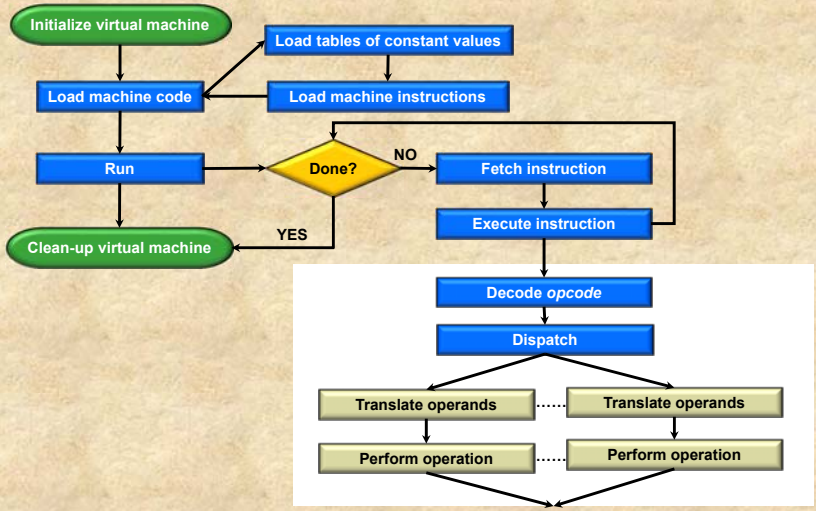
ΔΙΔΑΣΚΩΝ
Αντώνιος Σαββίδης

Διάλεξη 15η
ΚΑΤΑΣΚΕΥΗ ΕΙΚΟΝΙΚΗΣ ΜΗΧΑΝΗΣ – *μία διάλεξη*

Περιεχόμενα

- *Γενικό διάγραμμα ροής*
- Μετατροπή ορισμάτων
- Dispatcher
- Υλοποίηση εντολών
- Συναρτήσεις βιβλιοθήκης

Γενικό διάγραμμα ροής (1/1)



Περιεχόμενα

- Γενικό διάγραμμα ροής
- *Μετατροπή ορισμάτων*
- Dispatcher
- Υλοποίηση εντολών
- Συναρτήσεις βιβλιοθήκης

Μετατροπή ορισμάτων (1/5)

- Στις εντολές μηχανής εμφανίζονται αρκετοί διαφορετικοί τύποι ορισμάτων
  - Ορισμένα προσδιορίζουν κελιά μνήμης (μεταβλητές)
  - Άλλα σταθερές τιμές χρήστη (όπως ακέραιες τιμές ή strings constants)
  - Άλλα συναρτήσεις βιβλιοθήκης ή αυτές που ορίζονται στο πρόγραμμα
- Εάν η υλοποίηση των εντολών στην εικονική μηχανή χρησιμοποιεί αυτούσια τα ορίσματα όπως παρέχονται στις εντολές μηχανής θα πρέπει να ελέγχει όλες τις διαφορετικές περιπτώσεις ορισμάτων
  - Εφικτό μεν, αλλά κάνει πολύπλοκη την υλοποίηση κάθε εντολής
- Για την απλοποίηση αυτής της διαδικασίας θα υλοποιήσουμε έναν τρόπο όπου τα ορίσματα μετατρέπονται σε θέσεις μνήμης εισάγοντας τρεις νέους βοηθητικούς καταχωρητές
  - Σε αυτούς θα αποθηκεύουμε πάντα τα ορίσματα arg1 και arg2
  - Και σε ορισμένες εντολές το result

Μετατροπή ορισμάτων (2/5)

AX

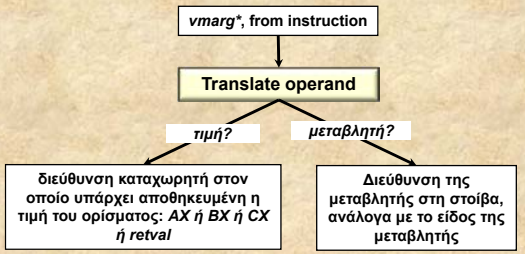
BX

CX

•Ο τύπος κάθε τέτοιου καταχωρητή είναι *avm_memcell*, δηλ. μπορεί να αποθηκεύσει τιμές της γλώσσας *alpha*.

• Όλοι αυτοί οι καταχωρητές χρησιμοποιούνται μόνο για operands που είναι τιμές, καθώς οι μεταβλητές είναι πάντα θέσεις μνήμης στη στοίβα.

•Ο σκοπός είναι να έχουμε στο τέλος εντολή με ορίσματα που είναι όλα *avm_memcell**, είτε από τη στοίβα είτε από κάποιον καταχωρητή. Αυτό θα διευκολύνει σημαντικά την υλοποίηση των επιμέρους εντολών για την εικονική μηχανή.
•Η μόνη περίπτωση των operands που δεν γίνονται translated είναι τα labels για τις εντολές JUMP.



Μετατροπή ορισμάτων (3/5)

```
#define AVM_STACKENV_SIZE   4
avm_memcell ax, bx, cx;
avm_memcell retval;
unsigned    top, topsp;

/*  Reverse translation for constants:
    getting constant value from index.
*/
double  consts_getnumber (unsigned index);
char*   consts_getstring (unsigned index);
char*   libfuncs_getused (unsigned index);
```

Η συνάρτηση αυτή θα καλείται όπως χρειάζεται στις συναρτήσεις που υλοποιούν τις εντολές της εικονικής μηχανής

```
avm_memcell* avm_translate_operand (vmarg* arg, avm_memcell* reg) {

    switch (arg->type) {

        /* Variables */
        case global_a:  return &stack[AVM_STACKSIZE-1-arg->val];
        case local_a:   return &stack[topsp-arg->val];
        case formal_a:  return &stack[topsp+AVM_STACKENV_SIZE+1+arg->val];

        case retval_a:  return &retval;
```

...συνεχίζεται

Environment function!!!

Οι παραπάνω τέσσερις (4) περιπτώσεις είναι οι μόνες που αντιπροσωπεύουν operands είναι μεταβλητές.

Μετατροπή ορισμάτων (4/5)

```
case number_a:  {
    reg->type = number_m;
    reg->data.numVal = consts_getnumber(arg->val);
    return reg;
}

case string_a:  {
    reg->type = string_m;
    reg->data.strVal = strdup(consts_getstring(arg->val));
    return reg;
}

case bool_a: {
    reg->type = bool_m;
    reg->data.boolVal = arg->val;
    return reg;
}

case nil_a: reg->type = nil_m; return reg;
```

•Για τις τιμές των ορισμάτων που αποθηκεύονται σε πίνακα σταθερών τιμών η εξαγωγή γίνεται χρησιμοποιώντας το operand value σαν index.

Μετατροπή ορισμάτων (5/5)

Για τις συναρτήσεις που ορίζει ο προγραμματιστής γνωρίζουμε ότι η διεύθυνση στον τελικό κώδικα μηχανής αποθηκεύεται απευθείας στην τιμή του ορίσματος

```
        case userfunc_a: {
            reg->type = userfunc_m;
            reg->data.funcVal = arg->val; /* Address already stored */
            return reg;
        }

        case libfunc_a: {
            reg->type = libfunc_m;
            reg->data.libfuncVal = libfuncs_getused(arg->val);
            return reg;
        }

        default: assert(0);
    }
}
```

Οι συναρτήσεις βιβλιοθήκης αντιμετωπίζονται ως σταθερές τιμές, με την τιμή του ορίσματος να είναι index στον πίνακα των ονομάτων των συναρτήσεων βιβλιοθήκης που χρησιμοποιούνται στο μεταγλωττισμένο πρόγραμμα.

Περιεχόμενα

- Γενικό διάγραμμα ροής
- Μετατροπή ορισμάτων
- *Dispatcher*
- Υλοποίηση εντολών
- Συναρτήσεις βιβλιοθήκης

Dispatcher (1/3)

- Ο κύκλος εκτέλεσης των εντολών της εικονικής μηχανής βασίζεται στην υλοποίηση ενός διανεμητή ο οποίος επιλέγει την κατάλληλη συνάρτηση εκτέλεσης εντολής ανάλογα με τον κώδικα εντολής
- Κάτι τέτοιο πρέπει να γίνεται γρήγορα, σε ένα βήμα, αποφεύγοντας *if ..else* εντολές ή *switches*
- Αντίστοιχη υλοποίηση έχουμε ήδη παρουσιάσει στην παραγωγή τελικού κώδικα βάσει του τύπου της εντολής ενδιάμεσου κώδικα
- Ο dispatcher καλεί τη σωστή εντολή εκτέλεσης, η οποία και εσωτερικά φροντίζει όπως απαιτείται την μετατροπή των χρησιμοποιούμενων ορισμάτων.

Dispatcher (2/3)

```
typedef void (*execute_func_t)(instruction*);

#define AVM_MAX_INSTRUCTIONS (unsigned) nop_v

extern void execute_assign (instruction*);
extern void execute_add (instruction*);
extern void execute_sub (instruction*);
extern void execute_mul (instruction*);
extern void execute_div (instruction*);
extern void execute_mod (instruction*);
extern void execute_uminus (instruction*);
extern void execute_and (instruction*);
extern void execute_or (instruction*);
extern void execute_not (instruction*);
extern void execute_jeq (instruction*);
extern void execute_jne (instruction*);
extern void execute_jle (instruction*);
extern void execute_jge (instruction*);
extern void execute_jlt (instruction*);
extern void execute_jgt (instruction*);
extern void execute_call (instruction*);
extern void execute_pusharg (instruction*);
extern void execute_funcenter (instruction*);
extern void execute_funcexit (instruction*);
extern void execute_newtable (instruction*);
extern void execute_tablegetelem (instruction*);
extern void execute_tablesetelem (instruction*);
extern void execute_nop (instruction*);
```

Καλό είναι να έχετε ξεχωριστά αρχεία με την υλοποίηση των διαφορετικών ομάδων εντολών

Dispatcher (3/3)

```
execute_func_t executeFuncs[]={
    execute_assign,
    execute_add,
    execute_sub,
    execute_mul,
    execute_div,
    execute_mod,
    execute_uminus,
    execute_and,
    execute_or,
    execute_not,
    execute_jeq,
    execute_jne,
    execute_jle,
    execute_jge,
    execute_jlt,
    execute_jgt,
    execute_call,
    execute_pusharg,
    execute_funcenter,
    execute_funcexit,
    execute_newtable,
    execute_tablegetelem,
    execute_tablesetelem,
    execute_nop
};
```

Προσοχή: η θέση ενός execute_<op> πρέπει να είναι ίδια με την αριθμητική τιμή του vmopcode για το <op>

```
unsigned char   executionFinished = 0;
unsigned        pc = 0;
unsigned        currLine = 0;
unsigned        codeSize = 0;
instruction*    code = (instruction*) 0;
#define AVM_ENDING_PC   codeSize

void execute_cycle (void) {
    if (executionFinished)
        return;
    else
    if (pc == AVM_ENDING_PC) {
        executionFinished = 1;
        return;
    }
    else {
        assert(pc < AVM_ENDING_PC);
        instruction* instr = code + pc;
        assert(
            instr->opcode >= 0 &&
            instr->opcode <= AVM_MAX_INSTRUCTIONS
        );
        if (instr->srcLine)
            currLine = instr->srcLine;
        unsigned oldPC = pc;
        (*executeFuncs[instr->opcode])(instr);
        if (pc == oldPC)
            ++pc;
    }
}
```

Το PC θα αλλάζει λόγω εντολών jump, call και επιστροφής από κλήση. Ειδάλλως, «πάμε» στην επόμενη εντολή.

Περιεχόμενα

- Γενικό διάγραμμα ροής
- Μετατροπή ορισμάτων
- Dispatcher
- *Υλοποίηση εντολών*
- Συναρτήσεις βιβλιοθήκης

Υλοποίηση εντολών (1/18)

- Πριν την υλοποίηση των εντολών πρέπει να παρουσιάσουμε μία βοηθητική συνάρτηση που, έχει ήδη αναφερθεί, για τον καθαρισμό ενός κελιού μνήμης.

```
typedef void (*memclear_func_t)(avm_memcell*);

extern void memclear_string (avm_memcell* m){
    assert(m->data.strVal);
    free(m->data.strVal);
}

extern void memclear_table (avm_memcell* m){
    assert(m->data.tableVal);
    avm_tabledecrefcounter(m->data.tableVal);
}
```

```
memclear_func_t memclearFuncs[]={
    0,  /* number */
    memclear_string,
    0,  /* bool */
    memclear_table,
    0,  /* userfunc */
    0,  /* livfunc */
    0,  /* nil */
    0   /* undef */
};

void avm_memcellclear (avm_memcell* m) {
    if (m->type != undef_m) {
        memclear_func_t f = memclearFuncs[m->type];
        if (f)
            (*f)(m);
        m->type = undef_m;
    }
}
```

- Όπως φαίνεται, ο καθαρισμός (collection) ενδιαφέρεται μόνο για strings και δυναμικούς πίνακες.
- Ακολουθούμε παντού τον τρόπο υλοποίησης με dispatch table τόσο για μεγαλύτερη ταχύτητα όσο και για καλύτερη κατάτμηση σε συναρτήσεις.
- *Αν δε θέλετε dispatch tables για κάποιο λόγο τότε γράψτε τα κλασικά switches.*

Υλοποίηση εντολών (2/18)

```
extern void avm_warning(char* format,...);
extern void avm_assign (avm_memcell* lv, avm_memcell* rv);

void execute_assign (instruction* instr) {

    avm_memcell* lv = avm_translate_operand(&instr->result, (avm_memcell*) 0);
    avm_memcell* rv = avm_translate_operand(&instr->arg1, &ax);

    assert(lv && ( &stack[N-1] >= lv && lv > &stack[top]  || lv==&retval));
    assert(rv); // should do similar assertion tests here

    avm_assign(lv,rv);
}
```

ASSIGN

•Προσοχή: το *lvalue*, δηλ. το result στην εντολή μηχανής, πρέπει απαραιτήτως να προκύψει ως stack cell και όχι κάτι πάνω από το παρόν activation record, ή έστω ως *retval register*.

•Επιπλέον, όταν γνωρίζω ότι ένα operand αντιπροσωπεύει ήδη μεταβλητή δεν περνάω ως argument τη διεύθυνση κάποιου καταχωρητή προς χρήση, αλλά *(avm_memcell*) 0*.

•Χρησιμοποιώ μία ξεχωριστή συνάρτηση *avm_assign* με ορίσματα κελιά μνήμης, αντί να γράψουμε τη λογική της εκχώρησης «χύμα», καθώς θα τη χρειαστούμε τόσο στους πίνακες όσο και στα activation records.

Υλοποίηση εντολών (3/18)

```
void avm_assign (avm_memcell* lv, avm_memcell* rv) {

    if (lv == rv)                /* Same cells? destructive to assign!*/
        return;

    if (lv->type == table_m &&  /* Same tables? no need to assign. */
        rv->type == table_m &&
        lv->data.tableVal == rv->data.tableVal)
        return;

    if (rv->type == undef_m)     /* From undefined r-value? warn! */
        avm_warning("assigning from 'undef' content!");

    avm_memcellclear(lv);        /* Clear old cell contents. */

    memcpy(lv, rv, sizeof(avm_memcell)); /* In C++ dispatch instead. */

    /* Now take care of copied values or reference counting.
    */
    if (lv->type == string_m)
        lv->data.strVal = strdup(rv->data.strVal);
    else
    if (lv->type == table_m)
        avm_tableincrefcounter(lv->data.tableVal);
}
```

ASSIGN

Κάθε κελί έχει το δικό του δυναμικό string, ενώ εάν εκχωρηθεί πίνακα φροντίζει να του αυξήσει το reference counter.

Υλοποίηση εντολών (4/18)

```
extern void  avm_error(char* format,...);
extern char* avm_tostring (avm_memcell*); /* Caller frees. */
extern void  avm_calllibfunc (char* funcName);
extern void  avm_callsaveenvironment (void);

void execute_call (instruction* instr) {

    avm_memcell* func = avm_translate_operand(&instr->result, &ax);
    assert(func);
    avm_callsaveenvironment();

    switch (func->type) {

        case userfunc_m : {
            pc = func->data.funcVal;
            assert(pc < AVM_ENDING_PC);
            assert(code[pc].opcode == funcenter_v);
            break;
        }

        case string_m:  avm_calllibfunc(func->data.strVal);     break;
        case libfunc_m: avm_calllibfunc(func->data.libfuncVal); break;

        default: {
            char* s = avm_tostring(func);
            avm_error("call: cannot bind '%s' to function!", s);
            free(s);
            executionFinished = 1;
        }
    }
}
```

CALLING FUNCTIONS

Ξέρουμε τα νόμιμα όρια διευθύνσεων εντολών, αλλά επίσης ξέρουμε ότι η πρώτη εντολή συνάρτησης είναι η *funcenter*.

Μία παράξενη ευκολία: μπορούμε να καλούμε Library functions και σαν strings!

Οτιδήποτε άλλο είναι runtime error και σηματοδοτεί μαζί με έναν ευγενές μήνυμα τον τερματισμό της εκτέλεσης.

Υλοποίηση εντολών (5/18)

```
unsigned totalActuals = 0;

void avm_dec_top (void) {
    if (!top) { /* Stack overflow */
        avm_error("stack overflow");
        executionFinished = 1;
    }
    else
        --top;
}

void avm_push_envvalue (unsigned val) {
    stack[top].type        = number_m;
    stack[top].data.numVal = val;
    avm_dec_top();
}

void avm_callsaveenvironment (void) {
    avm_push_envvalue(totalActuals);
    avm_push_envvalue(pc+1);
    avm_push_envvalue(top + totalActuals + 2);
    avm_push_envvalue(topsp);
}
```

CALLING FUNCTIONS

Προσέχουμε και ελέγχουμε πάντα για το ενδεχόμενο ενός stack overflow.

•Η ακολουθία κλήσης ολοκληρώνεται με την αποθήκευση του περιβάλλοντος.

• Προσοχή στην αριθμητική των αποθηκευμένων τιμών (να γίνει έλεγχος χειροκίνητα).

Υλοποίηση εντολών (6/18)

```
extern userfunc* avm_getfuncinfo (unsigned address);

void execute_funcenter (instruction* instr) {
    avm_memcell* func = avm_translate_operand(&instr->result, &ax);
    assert(func);
    assert(pc == func->data.funcVal); /* Func address should match PC. */

    /* Callee actions here. */
    totalActuals = 0;
    userfunc* funcInfo = avm_getfuncinfo(pc);
    topsp = top;
    top = top - funcInfo->localSize;
}
```

CALLING FUNCTIONS

```
unsigned avm_get_envvalue (unsigned i) {
    assert(stack[i].type = number_m);
    unsigned val = (unsigned) stack[i].data.numVal;
    assert(stack[i].data.numVal == ((double) val));
    return val;
}

void execute_funcexit (instruction* unused) {
    unsigned oldTop = top;
    top     = avm_get_envvalue(topsp + AVM_SAVEDTOP_OFFSET);
    pc      = avm_get_envvalue(topsp + AVM_SAVEDPC_OFFSET);
    topsp   = avm_get_envvalue(topsp + AVM_SAVEDTOPSP_OFFSET);

    while (++oldTop <= top) /* Intentionally ignoring first. */
        avm_memcellclear(&stack[oldTop]);
}
```

```
#define AVM_NUMACTUALS_OFFSET   +4
#define AVM_SAVEDPC_OFFSET      +3
#define AVM_SAVEDTOP_OFFSET     +2
#define AVM_SAVEDTOPSP_OFFSET   +1
```

Επαναφορά προηγούμενου περιβάλλοντος και επιστροφή από την κλήση.

Καθαρισμός του activation record (garbage collection).

Υλοποίηση εντολών (7/18)

```
typedef void (*library_func_t)(void);
library_func_t avm_getlibraryfunc (char* id); /* Typical hashing. */

void avm_calllibfunc (char* id) {
    library_func_t f = avm_getlibraryfunc(id);
    if (!f) {
        avm_error("unsupported lib func '%s' called!", id);
        executionFinished = 1;
    }
    else {
        /*  Notice that enter function and exit function
            are called manually!
        */
        topsp = top;    /* Enter function sequence. No stack locals. */
        totalActuals = 0;
        (*f)();         /* Call library function. */
        if (!executionFinished) /* An error may naturally occur inside. */
            execute_funcexit((instruction*) 0); /* Return sequence. */
    }
}
```

CALLING FUNCTIONS

Καθώς οι συναρτήσεις βιβλιοθήκης είναι υλοποιημένες σε C/C++, πρέπει να καλέσουμε χειροκίνητα την ακολουθία εντολών που οφείλει να εκτελέσει μία συνάρτηση όταν αρχίζει να εκτελείται, καθώς και τις εντολές της ακολουθίας εξόδου. Προσέξτε ότι οι συναρτήσεις βιβλιοθήκης δεν περιέχουν τοπικές μεταβλητές στοίβας της εικονικής μηχανής (μπορούν βεβαίως να έχουν τοπικές μεταβλητές του προγράμματος C/C++).

Υλοποίηση εντολών (8/18)

```
unsigned avm_totalactuals (void) {
    return avm_get_envvalue(topsp + AVM_NUMACTUALS_OFFSET);
}

avm_memcell* avm_getactual (unsigned i) {
    assert(i < avm_totalactuals());
    return &stack[topsp + AVM_STACKENV_SIZE + 1 + i];
}

/*  Implementation of the library function 'print'.
    It displays every argument at the console.
*/
void libfunc_print (void) {
    unsigned n = avm_totalactuals();
    for (unsigned i = 0; i < n; ++i) {
        char* s = avm_tostring(avm_getactual(i));
        puts(s);
        free(s);
    }
}

/*  With the following every library function is manually
    added in the VM library function resolution map.
*/
void avm_registerlibfunc (char* id, library_func_t addr);
```

CALLING FUNCTIONS

•Πως υλοποιούμε τις συναρτήσεις βιβλιοθήκης; Ως απλές C/C++ συναρτήσεις οι οποίες λαμβάνουν τα πραγματικά ορίσματα από τη στοίβα της εικονικής μηχανής.

•Το ίδιο ισχύει και στις συναρτήσεις βιβλιοθήκης για γλώσσες γενικού σκοπού και παραγωγή κώδικα για πραγματικές μηχανές.

•Καθώς ο τρόπος χρήσης της στοίβας και των καταχωρητών είναι θέμα του compiler, η υλοποίηση συναρτήσεων βιβλιοθήκης για έναν compiler δεν είναι de facto portable και σε έναν άλλον compiler της ίδιας γλώσσας και για την ίδια μηχανή.

Υλοποίηση εντολών (9/18)

```
void execute_pusharg (instruction* instr) {
    avm_memcell* arg = avm_translate_operand(&instr->arg1, &ax);
    assert(arg);

    /*  This is actually stack[top] = arg, but we have to
        use avm_assign.
    */
    avm_assign(&stack[top], arg);
    ++totalActuals;
    avm_dec_top();
}
```

CALLING FUNCTIONS

```
typedef char* (*tostring_func_t)(avm_memcell*);

extern char* number_tostring (avm_memcell*);
extern char* string_tostring (avm_memcell*);
extern char* bool_tostring (avm_memcell*);
extern char* table_tostring (avm_memcell*);
extern char* userfunc_tostring (avm_memcell*);
extern char* libfunc_tostring (avm_memcell*);
extern char* nil_tostring (avm_memcell*);
extern char* undef_tostring (avm_memcell*);

tostring_func_t tostringFuncs[]={
    number_tostring,
    string_tostring,
    bool_tostring,
    table_tostring,
    userfunc_tostring,
    libfunc_tostring,
    nil_tostring,
    undef_tostring
};
char* avm_tostring (avm_memcell* m) {
    assert(m->type >= 0 && m->type <= undef_m);
    return (*tostringFuncs[m->type])(m);
}
```

•Ο τρόπος προώθησης ενός πραγματικού ορίσματος στη στοίβα είναι τετριμμένος.

•Οι υλοποίηση των επιμέρους συναρτήσεων για μετατροπή ενός κελιού σε string είναι τετριμμένη.

•Προσοχή στους υπολογισμούς μεγεθών των buffers για τη δημιουργία δυναμικού string (εάν χρησιμοποιήσετε *sprintf*).

Υλοποίηση εντολών (10/18)

```
typedef double (*arithmetic_func_t)(double x, double y);

double add_impl (double x, double y) { return x+y; }
double sub_impl (double x, double y) { return x-y; }
double mul_impl (double x, double y) { return x*y; }
double div_impl (double x, double y) { return x/y; /* Error check? */ }
double mod_impl (double x, double y) {
    return ((unsigned) x) % ((unsigned) y); /* Error check? */
}

/* Dispatcher just for arithmetic functions. */
arithmetic_func_t arithmeticFuncs[] = {
    add_impl,
    sub_impl,
    mul_impl,
    div_impl,
    mod_impl
};
```

```
#define execute_add execute_arithmetic
#define execute_sub execute_arithmetic
#define execute_mul execute_arithmetic
#define execute_div execute_arithmetic
#define execute_mod execute_arithmetic
```

```
void execute_arithmetic (instruction* instr) {

    avm_memcell* lv  = avm_translate_operand(&instr->result, (avm_memcell*) 0);
    avm_memcell* rv1 = avm_translate_operand(&instr->arg1, &ax);
    avm_memcell* rv2 = avm_translate_operand(&instr->arg2, &bx);

    assert(lv && ( &stack[N-1] >= lv && lv > &stack[top] || lv==&retval));
    assert(rv1 && rv2);

    if (rv1->type != number_m || rv2->type != number_m) {
        avm_error("not a number in arithmetic!");
        executionFinished = 1;
    }
    else {
        arithmetic_func_t op = arithmeticFuncs[instr->opcode - add_v];
        avm_memcellclear(lv);
        lv->type        = number_m;
        lv->data.numVal = (*op)(rv1->data.numVal, rv2->data.numVal);
    }
}
```

ARITHMETIC OPERATIONS

•Η υλοποίηση των αριθμητικών εντολών. Μην ξεχάσετε τους ελέγχους για runtime error.

Υλοποίηση εντολών (11/18)

- Με τρόπο παρόμοιο των αριθμητικών εκφράσεων υλοποιούνται και οι συσχετιστικοί τελεστές διάταξης < <= > >=, δηλ. οι εντολές JGE, JGT, JLE, JLT, καθώς αφορούν μόνο αριθμούς.
- Προσοχή θέλει το γεγονός ότι δεν χρειάζεται να μετατρέψουμε το operand στο οποίο είναι αποθηκευμένη η διεύθυνση (label) της εντολής προορισμού
- Οι βοηθητικές συναρτήσεις **`comparisonFuncs`** θα είναι έχουν αντίστοιχο signature, δηλ.
  - **`bool (*cmp_func) (double, double)`**

Υλοποίηση εντολών (12/18)

- Οι εντολές συσχετιστικών τελεστών ισότητας έχουν διαφορετική υλοποίηση ώστε να ικανοποιούνται οι σημασιολογικοί κανόνες της γλώσσας
  - Σύγκριση με undefined προκαλεί *runtime error,*
  - αλλιώς οτιδήποτε είναι συγκρίσιμο == με nil και το αποτέλεσμα είναι true μόνο εάν και τα δύο nil
  - αλλιώς σύγκριση με boolean απαιτεί μετατροπή σε boolean τιμή
  - αλλιώς η σύγκριση επιτρέπεται μόνο μεταξύ ομοειδών (ιδίου τύπου)

Υλοποίηση εντολών (13/18)

```
typedef unsigned char (*tobool_func_t)(avm_memcell*);

unsigned char number_tobool (avm_memcell* m) { return m->data.numVal != 0; }
unsigned char string_tobool (avm_memcell* m) { return m->data.strVal[0] != 0; }
unsigned char bool_tobool (avm_memcell* m) { return m->data.boolVal; }
unsigned char table_tobool (avm_memcell* m) { return 1; }
unsigned char userfunc_tobool (avm_memcell* m) { return 1; }
unsigned char libfunc_tobool (avm_memcell* m) { return 1; }
unsigned char nil_tobool (avm_memcell* m) { return 0; }
unsigned char undef_tobool (avm_memcell* m) { assert(0); return 0;  }

tobool_func_t toboolFuncs[]={
    number_tobool,
    string_tobool,
    bool_tobool,
    table_tobool,
    userfunc_tobool,
    libfunc_tobool,
    nil_tobool,
    undef_tobool
};

unsigned char avm_tobool (avm_memcell* m) {
    assert(m->type >= 0 && m->type < undef_m);
    return (*toboolFuncs[m->type])(m);
}
```

Η μετατροπή σε boolean θα μας χρειαστεί και καλό είναι να έχουμε ταχύτατη (ως προς την εκτέλεση) υλοποίηση.

EQUALITY OPERATIONS

Υλοποίηση εντολών (14/18)

```
void execute_jeq (instruction* instr) {

    assert(instr->result.type == label_a);

    avm_memcell* rv1 = avm_translate_operand(&instr->arg1, &ax);
    avm_memcell* rv2 = avm_translate_operand(&instr->arg2, &bx);

    unsigned char result = 0;

    if (rv1->type == undef_m || rv2->type == undef_m)
        avm_error("'undef' involved in equality!");
    else
    if (rv1->type == nil_m || rv2->type == nil_m)
        result =  rv1->type == nil_m && rv2->type == nil_m;
    else
    if (rv1->type == bool_m || rv2->type == bool_m)
        result = (avm_tobool(rv1) == avm_tobool(rv2));
    else
    if (rv1->type != rv2->type)
        avm_error(
            "%s == %s is illegal!",
            typeStrings[rv1->type],
            typeStrings[rv2->type]
        );
    else {
        /* Equality check with dispatching. */
    }

    if(!executionFinished && result)
        pc = instr->result.val;
}
```

EQUALITY OPERATIONS

```
char* typeStrings[]={
    "number",
    "string",
    "bool",
    "table",
    "userfunc",
    "libfunc",
    "nil",
    "undef"
};
```

Θεωρούμε ότι ενσωματώνουμε στην *avm_error* και την εντολή *executionFinished*=1

Εδώ συμπληρώστε την υλοποίηση. Αρκεί να κάνετε dispatching ως προς τον τύπο του *rv1*

Εάν δεν είχαμε runtime error και το αποτέλεσμα είναι *true*, πρέπει να εφαρμοστεί το *jump* (δηλ. να θέσουμε το PC)

Υλοποίηση εντολών (15/18)

- Εδώ παρουσιάζουμε και την υλοποίηση της *typeof* συνάρτησης βιβλιοθήκης, η οποία επιτρέπει runtime type identification (ελέγχει ακόμη και undefined variables).
- Στην περίπτωση που μίας συνάρτηση βιβλιοθήκης χρειάζεται συγκεκριμένο αριθμό από arguments, προφανώς υλοποιεί εσωτερικά και τη λογική έλεγχου και ανάλογα μπορεί να εξάγει ένα *runtime error*.
- Επιπλέον φαίνεται ο τρόπος με τον οποίο υλοποιούμε συναρτήσεις βιβλιοθήκης οι οποίες επιστρέφουν τιμές (απλώς θέτουν τον *retval* register).

```
void libfunc_typeof (void) {

    unsigned n = avm_totalactuals();

    if (n !=1)
        avm_error("one argument (not %d) expected in 'typeof'!", n);
    else {

        /*  Thats how a library function returns a result.
            It has to only set the 'retval' register!
        */
        avm_memcellclear(&retval); /* Don't forget to clean-it up! */
        retval.type = string_m;
        retval.data.strVal = strdup(typeStrings[avm_getactual(0)->type]);
    }
}
```

LIBRARY FUNCTIONS

Υλοποίηση εντολών (16/18)

Οι τιμές των πινάκων γνωρίζουμε ότι πάντα αποθηκεύονται σε κάποια μεταβλητή. Επομένως δεν χρειαζόμαστε βοηθητικό καταχωρητή.

```
void execute_newtable (instruction* instr) {
    avm_memcell* lv  = avm_translate_operand(&instr->result, (avm_memcell*) 0);
    assert(lv && ( &stack[N-1] >= lv && lv > &stack[top]  || lv==&retval));

    avm_memcellclear(lv);

    lv->type            = table_m;
    lv->data.tableVal   = avm_tablenew();
    avm_tableincrefcounter(lv->data.tableVal);
}

avm_memcell*    avm_tablegetelem (
                    avm_table*      table,
                    avm_memcell*    index
                );

void            avm_tablesetelem (
                    avm_table*      table,
                    avm_memcell*    index,
                    avm_memcell*    content

                );
```

Δεν ξεχνάμε την αύξηση του μετρητή αναφορών, καθώς κατά τη δημιουργία ο πίνακας έχει τον μετρητή αυτόν στην τιμή 0.

TABLES

Υλοποίηση εντολών (17/18)

```
void execute_tablegetelem (instruction* instr) {

    avm_memcell* lv  = avm_translate_operand(&instr->result, (avm_memcell*) 0);
    avm_memcell* t   = avm_translate_operand(&instr->arg1, (avm_memcell*) 0);
    avm_memcell* i   = avm_translate_operand(&instr->arg2, &ax);

    assert(lv &&   &stack[N-1] >= lv && lv > &stack[top]   || lv==&retval));
    assert(t &&  &stack[N-1] >= t && t > &stack[top] );
    assert(i);

    avm_memcellclear(lv);
    lv->type = nil_m;   /* Default value. */

    if (t->type != table_m) {
        avm_error("illegal use of type %s as table!", typeStrings[t->type]);
    }
    else {
        avm_memcell* content = avm_tablegetelem(t->data.tableVal, i);
        if (content)
            avm_assign(lv, content);
        else {
            char* ts = avm_tostring(t);
            char* is = avm_tostring(i);
            avm_warning("%s[%s] not found!", ts, is);
            free(ts);
            free(is);
        }
    }
}
```

Ενδέχεται το στοιχείο που ζητείται για το συγκεκριμένο κλειδί απλώς να μην υπάρχει ή να μην υποστηρίζεται κλειδί του συγκεκριμένου τύπου.

TABLES

Υλοποίηση εντολών (18/18)

- Η υλοποίηση της ***execute_tablesetelem*** καθαυτού είναι απλή, αλλά βασίζεται στην ***avm_talesetelem*** η οποία είναι κατασκευαστικά αρκετά απαιτητική
  - Να μην ξεχάσετε ότι με *nil* index αφαιρείται το στοιχείο, δηλ. το *nil* δεν μπορεί να αποθηκεύεται σε πίνακες
  - και ότι πρέπει να χρησιμοποιείτε την ***avm_assign*** και ***avm_memcellclear***

```
void execute_tablesetelem (instruction* instr) {
    avm_memcell* t   = avm_translate_operand(&instr->result, (avm_memcell*) 0);
    avm_memcell* i   = avm_translate_operand(&instr->arg1, &ax);
    avm_memcell* c   = avm_translate_operand(&instr->arg2, &bx);

    assert(t && &stack[N-1] >= t && t > &stack[top] );
    assert(i && c);

    if (t->type != table_m)
        avm_error("illegal use of type %s as table!", typeStrings[t->type]);
    else
        avm_tablesetelem(t->data.tableVal, i, c);
}
```

Γιατί δεν ελέγχουμε και για *retval* register?

Περιεχόμενα

- Γενικό διάγραμμα ροής
- Μετατροπή ορισμάτων
- Dispatcher
- Υλοποίηση εντολών
- *Συναρτήσεις βιβλιοθήκης*

Συναρτήσεις βιβλιοθήκης (1/2)

- Έχουμε δει τον τρόπο υλοποίησης τους, καθώς ξεκαθάρισαν δύο βασικά θέματα:
  - πως λαμβάνουμε τα πραγματικά ορίσματα από τη στοίβα της εικονικής μηχανής
  - πως επιστρέφουμε κάποιο αποτέλεσμα στην εικονική μηχανή, με τρόπο ίδιο με τις συναρτήσεις του προγράμματος
- Μένουν δύο ακόμη λεπτομέρειες
  - Η οργάνωση των συναρτήσεων
  - Και η υλοποίηση ειδικών συναρτήσεων

```
void avm_initialize (void) {

    avm_initstack();

    avm_registerlibfunc("print", libfunc_print);
    avm_registerlibfunc("typeof", libfunc_typeof);

    /* Same for all the rest library functions.
    */
}
```

Στην αρχικοποίηση της εικονικής μηχανής γίνεται και το installation των βιβλιοθηκών. Αυτό το μοντέλο είναι αντίστοιχο με το να έχουμε *statically linked library.*

Συναρτήσεις βιβλιοθήκης (2/2)

activation record της κλήσης *totalarguments*
activation record του caller της *totalarguments*
...
...
....

```
function f() {
    n = totalarguments();
}
```

•Όταν κληθεί η *totalarguments*, στην υλοποίηση της σε C/C++ η κλήση *avm_totalactuals*() επιστρέφει τον αριθμό των arguments στην ίδια την συνάρτηση και όχι στον caller αυτής όπως θα έπρεπε.
•Επομένως για να επιστραφεί ο σωστός αριθμός πρέπει να κινηθούμε ένα activation record κάτω.

```
void libfunc_totalarguments (void) {

    /* Get topsp of previous activation record.
    */
    unsigned p_topsp = avm_get_envvalue(topsp + AVM_SAVEDTOPSP_OFFSET);
    avm_memcellclear(&retval);

    if (!p_topsp) { /* If 0, no previous activation record. */
        avm_error("'totalarguments' called outside a function!");
        retval.type = nil_m;
    }
    else {
        /* Extract the number of actual arguments for the previous
           activation record. */
        retval.type = number_m;
        retval.data.numVal = avm_get_envvalue(p_topsp + AVM_NUMACTUALS_OFFSET);
    }
}
```

Με παρόμοιο τρόπο υλοποιήστε την *argument*(i)