Házi feladat
Programozás alapjai 2
Végleges

Készítette: Varga Zsófi

Tartalomjegyzék

1. Feladat

Készítsen objektummodellt egyszerű Neumann-elvű számítógép modellezéséhez! Definiáljon utasítás-objektumokat, melyek a memória-objektumban tárolhatók. A vezérlő egység feladata a memóriából kivenni a soron következő utasítás objektumot és aktivizálni ill. meghatározni a következő utasítás helyét.
Demonstrálja a működést egy egyszerű algoritmus megvalósításával (pl Fibonacci sorozat)! Nem kell a gép összes utasítását megvalósítani. Elegendő azokat, amit a demonstrációban felhasznál. A megoldáshoz ne használjon STL tárolót!

2. Pontosított specifikáció

2.1 Felhasználói interakció

A felhasználó a számítógéppel terminálon (cout/cin) keresztül kommunikál.
Indításkor a számítógép egy szöveges fájl-t kér a felhasználótól, ami a végrehajtandó utasításokat tartalmazza.

2.2 Program bevitele

A bemeneti fájl sorai az utasításokat és azok memóriacímét tartalmazzák, illetve az első sor tartalmazza, hogy mekkora legyen a memória. Példa program kód, ami kiszámolja 1+2 értékét és kiírja cout-ra:

ADDR	INSTR	OPERAND
0x0000	LOAD	0x0101
0x0001	ADD	0x0102
0x0002	STORE	0x0103
0x0003	PRINT	0x0103
0x0101	VAR	0x0001
0x0102	VAR	0x0002

A számítógép a fájl tartalmával inicializálja a memóriáját és elkezdi a program végrehajtását.

2.3 Lehetséges instrukciók

A számítógép a végrehajtást a 0x0000 memóriacímről kezdi. A következő instrukciók lehetségesek:
LOAD OPERAND -> betölti az OPERAND címen lévő var-t az ACC regiszterbe.
STORE OPERAND -> elmenti az ACC register tartalmát a OPERAND címre var-ként
ADD OPERAND -> hozzáadja az ACC regiszterben lévő értékhez az OPERAND címen lévő értéket.
SUB OPERAND -> kivonja az ACC registerben lévő értékből az OPERAND címen lévő értéket

READ OPERAND -> beolvas az input-ról (std::cin) egy számot és eltárolja az OPERAND címen var-ként
PRINT OPERAND -> kiírja az OPERAND címen lévő var-t az output-ra (std>>cout)
BRANCHGT OPERAND -> a következő utasítás címe OPERAND-ra változik, ha ACC > 0
JUMP OPERAND -> a következő utasítás címe OPERAND-ra változik

2.4 Adat deklarálása

Annak érdekében, hogy az instrukciókat és konstansokat egy tárolóban tárolhassam a konstansokat speciális instrukciókként kezelem.
Az input file-ban ezek VAR instrukcióval vannak jelölve. A VAR instrukció esetén az operand lehet 32 bit.

2.5 Hibakezelés

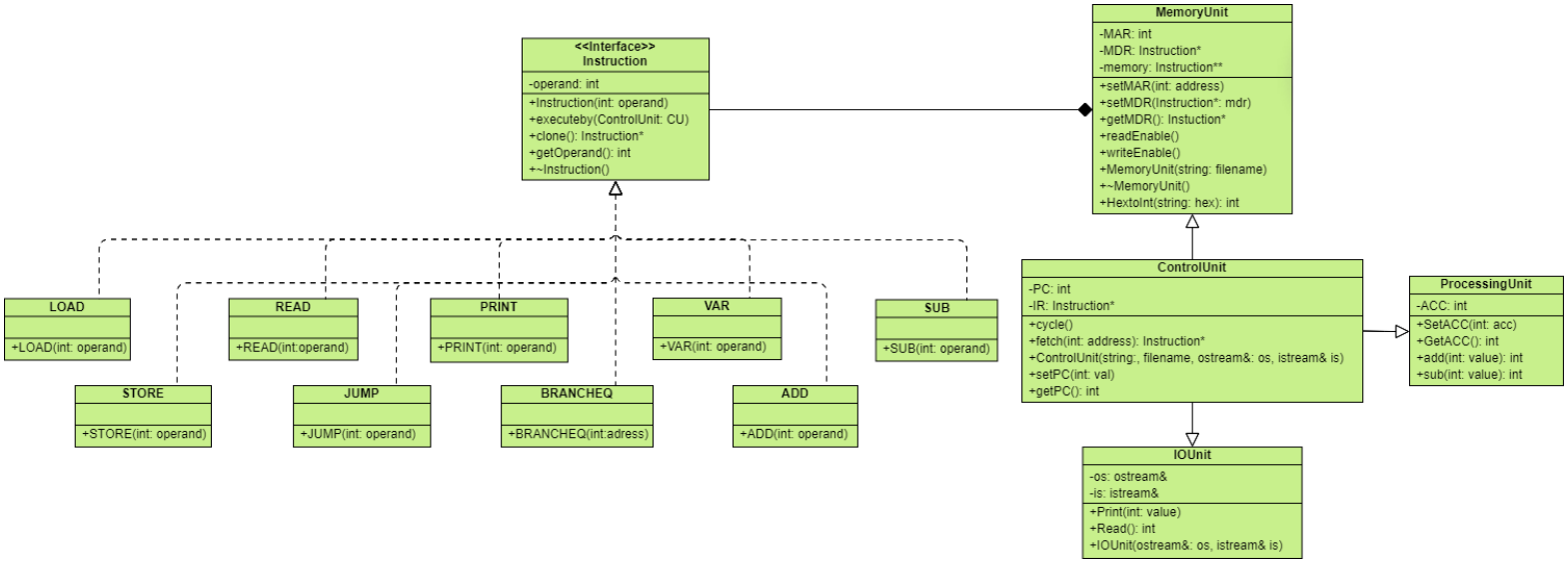
Amennyiben egy VAR instrukciót akarunk végrehajtani az dobni fog egy kivételt.

2.6 Demonstrációs program

A számítógép működésének demostrációjához egy szöveges fájlban egy programot fogok adni, ami kiszámolja az n-ik fibonacci számot.

3. Osztályok megvalósítása

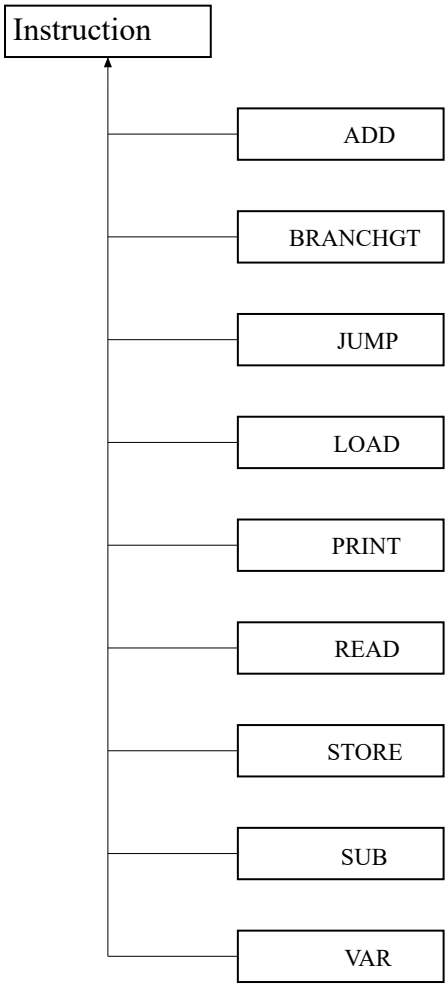
3.1 Osztálydiagram



3.2 **Instruction** absztrakt osztály

Az egyes kódok megírásához az Instruction leszármazott osztályait kell használni. Mindegyik leszármazott osztálynak külön feladata (a fentebb leírtak szerint) van, ezek a feladatok az executeby függvényekben van leimplementálva.

Az Instruction osztály származási diagramja:



Privát adattagok:

- int operand -> instrukció címe, vagy VAR instrukciónál a konstans értéke.

Publikus tagfüggvények:

- Instruction (int operand)
- int getOperand ()
- virtual void executeby (ControlUnit &CU)=0
- virtual Instruction ∗ clone ()=0
- virtual ∼Instruction ()

3.2.1 Konstruktor és destruktor dokumentációja

Instruction::Instruction(int operand) [inline]

Konstruktor

- *Paraméterek:*
  - **operand** -> cím vagy konstans

Instruction::~Instruction() [virtual]

Destruktor

3.2.2 Tagfüggvények dokumentációja

int Instruction::getOperand() [inline]

Operand lekérdezése

- *Visszatérési érték:*
  - instrukció címe vagy konstans

void executeby(ControlUnit& CU) [virtual]

Végrehajtja a megfelelő instrukcióhoz tartozó utasítást.

- *Paraméterek:*
  - **CU** -> Vezérlőegység, amin el kell végezni az instrukciót

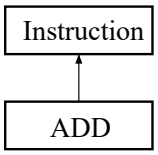
Instruction* clone() [virtual]

Létrehoz egy másolatot dinamikusan.

- *Visszatérési érték:*
  - a létrehozott dinamikus objektumra mutató pointer

3.3 **ADD** osztály

#include <Instruction.h>
Az ADD osztály származási diagramja:



Publikus tagfüggvények:

- ADD (int operand)
- void executeby (ControlUnit &CU)
- Instruction * clone ()
- ~ADD ()

3.3.1 Konstruktor és destruktor dokumentációja

ADD::ADD(int operand) [inline]

ADD osztály konstruktora, meghívja az Instrukció osztály konstruktorát az operanddal.

- *Paraméterek:*
  - **operand** -> a konstans címe, amit hozzá kell majd adni az akkumlátorhoz

ADD::~ADD() [inline]

Destruktor

3.3.2 Tagfüggvények dokumentációja

Instruction ∗ ADD::clone ( )

Létrehoz egy dinamikus ADD példányt.

- *Visszatérési érték:*
  - Létrehozott példányra mutató pointer

void executeby(ControlUnit& CU)

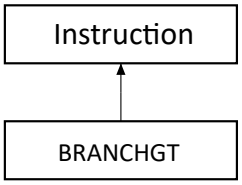
Hozzáadja az operand címen lévő konstanst az akkumlátorhoz.

- *Paraméterek:*
  - **CU** -> Vezérlőegység

3.4 **BRANCHGT** osztály

#include <Instruction.h>

A BRANCHGT osztály származási diagramja:



Publikus tagfüggvények

- BRANCHGT (int operand)
- void executeby (ControlUnit &CU)
- Instruction ∗ clone ()
- ∼BRANCHGT ()

3.4.1 Konstruktor és destruktor dokumentációja

BRANCHGT:: BRANCHGT (int operand) [inline]

BRANCHGT osztály konstruktora, meghívja az Instrukció osztály konstruktorát az operanddal.

- *Paraméterek:*
  - **operand** -> a következőnek elvégezendő utasítás címe

BRANCHGT::~ BRANCHGT () [inline]

Destruktor

3.4.2 Tagfüggvények dokumentációja

Instruction ∗ BRANCHGT::clone ( )

Létrehoz egy dinamikus BRANCHGT példányt.

- *Visszatérési érték:*
  - Létrehozott példányra mutató pointer

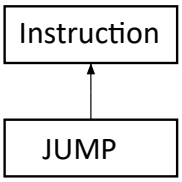
void executeby(ControlUnit& CU)

Ha az akkumlátor értéke nagyobb, mint nulla akkor a Programszámláló értéke operandra változik. Ha az ugrási cím kisebb, mint nulla vagy nagyobb, mint a memória kapacitása, akkor kivételt dob.

- *Paraméterek:*
  - **CU** -> Vezérlőegység

3.5 **JUMP** osztály

#include <Instruction.h>
A JUMP osztály származási diagramja:



Publikus tagfüggvények

- JUMP (int operand)
- void executeby (ControlUnit &CU)
- Instruction ∗ clone ()
- ~JUMP ()

3.5.1 Konstruktor és destruktor dokumentációja

JUMP:: JUMP (int operand) [inline]

JUMP osztály konstruktora, meghívja az Instrukció osztály konstruktorát az operanddal.

- *Paraméterek:*
  - **operand** -> a következőnek elvégezendő utasítás címe

JUMP::~ JUMP () [inline]

Destruktor

3.5.2 Tagfüggvények dokumentációja

Instruction ∗JUMP::clone ( )

Létrehoz egy dinamikus JUMP példányt.

- *Visszatérési érték:*
  - Létrehozott példányra mutató pointer

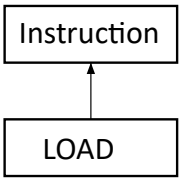
void executeby(ControlUnit& CU)

Átállítja a Programszámláló (PC) értékét az operandra. Ha az ugrási cím kisebb, mint nulla vagy nagyobb, mint a memória kapacitása, akkor kivételt dob.

- *Paraméterek:*
  - **CU** -> Vezérlőegység

3.6 **LOAD** osztály

#include <Instruction.h>
A LOAD osztály származási diagramja:



Publikus tagfüggvények

- LOAD (int operand)
- void executeby (ControlUnit &CU)
- Instruction ∗ clone ()
- ∼LOAD ()

3.6.1 Konstruktor és destruktor dokumentációja

LOAD:: LOAD(int operand) [inline]

LOAD osztály konstruktora, meghívja az Instrukció osztály konstruktorát az operanddal.

- *Paraméterek:*
  - **operand** -> cím, ahonnan betölti majd betölti az adatot

LOAD::~LOAD() [inline]

Destruktor

3.6.2 Tagfüggvények dokumentációja

Instruction ∗LOAD::clone ( )

Létrehoz egy dinamikus LOAD példányt.

- *Visszatérési érték:*
  - Létrehozott példányra mutató pointer

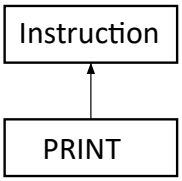
void executeby(ControlUnit& CU)

Lekéri a Vezérlőegységtől az operand címen lévő operandus értékét, majd berakja az akumlátorba.

- *Paraméterek:*
  - **CU** -> Vezérlőegység

3.7 **PRINT** osztály

#include <Instruction.h>
A PRINT osztály származási diagramja:



Publikus tagfüggvények

- PRINT (int operand)
- void executeby (ControlUnit &CU)
- Instruction ∗ clone ()
- ∼ PRINT ()

3.7.1 Konstruktor és destruktor dokumentációja

PRINT:: PRINT (int operand) [inline]

PRINT osztály konstruktora, meghívja az Instrukció osztály konstruktorát az operanddal.

- *Paraméterek:*
  - **operand** -> cím, aminek az értéket ki kell írni

PRINT::~ PRINT() [inline]

Destruktor

3.7.2 Tagfüggvények dokumentációja

Instruction ∗ PRINT::clone ( )

Létrehoz egy dinamikus PRINT példányt.

- *Visszatérési érték:*
  - Létrehozott példányra mutató pointer

void executeby(ControlUnit& CU)

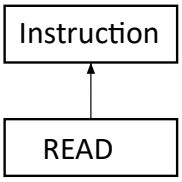
Kiolvassa az operand címen lévő adatot, majd kiírja az értéket.

- *Paraméterek:*
  - **CU** -> Vezérlőegység

3.8 **READ** osztály

#include <Instruction.h>

A READ osztály származási diagramja:



Publikus tagfüggvények

- READ (int operand)
- void executeby (ControlUnit &CU)
- Instruction ∗ clone ()
- ∼ READ ()

3.8.1 Konstruktor és destruktor dokumentációja

READ:: READ(int operand) [inline]

READ osztály konstruktora, meghívja az Instrukció osztály konstruktorát az operanddal.

- *Paraméterek:*
  - **operand** -> cím, ahova el menti az adatot

READ::~ READ() [inline]

Destruktor

3.8.2 Tagfüggvények dokumentációja

Instruction ∗READ::clone ( )

Létrehoz egy dinamikus READ példányt.

- *Visszatérési érték:*
  - Létrehozott példányra mutató pointer

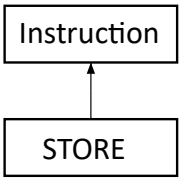
void executeby(ControlUnit& CU)

Beolvassa a bemeneti értéket majd elmenti az operand címre konstansként.

- *Paraméterek:*
  - **CU** -> Vezérlőegység

3.9 **STORE** osztály

#include <Instruction.h>
A STORE osztály származási diagramja:



Publikus tagfüggvények

- STORE (int operand)
- void executeby (ControlUnit &CU)
- Instruction ∗ clone ()
- ∼ STORE ()

3.9.1 Konstruktor és destruktor dokumentációja

STORE::STORE(int operand) [inline]

STORE osztály konstruktora, meghívja az Instrukció osztály konstruktorát az operanddal.

- *Paraméterek:*
  - **operand** -> cím, ahova el kell tárolni az adatot

STORE::~ STORE() [inline]

Destruktor

3.9.2 Tagfüggvények dokumentációja

Instruction ∗STORE::clone ( )

Létrehoz egy dinamikus STORE példányt.

- *Visszatérési érték:*
  - Létrehozott példányra mutató pointer

void executeby(ControlUnit& CU)

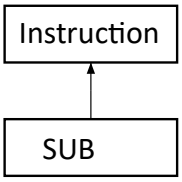
Beolvassa a bemeneti értéket majd elmenti az operand címre konstansként.

- *Paraméterek:*
  - **CU** -> Vezérlőegység

3.10 **SUB** osztály

#include <Instruction.h>

A SUB osztály származási diagramja:



Publikus tagfüggvények

- SUB (int operand)
- void executeby (ControlUnit &CU)
- Instruction ∗ clone ()
- ∼ SUB ()

3.10.1 Konstruktor és destruktor dokumentációja

SUB::SUB(int operand) [inline]

SUB osztály konstruktora, meghívja az Instrukció osztály konstruktorát az operanddal.

- *Paraméterek:*
  - **operand** -> a konstans címe, amit ki kell majd vonni az akkumlátorból

SUB::~SUB() [inline]

Destruktor

3.10.2 Tagfüggvények dokumentációja

Instruction *SUB::clone ()

Létrehoz egy dinamikus SUB példányt.

- *Visszatérési érték:*
  - Létrehozott példányra mutató pointer

void executeby(ControlUnit& CU)

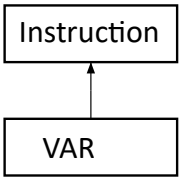
Kivonja az operand címen lévő konstanst az akkumlátorból.

- *Paraméterek:*
  - **CU** -> Vezérlőegység

3.11 **VAR** osztály

Ez az osztály egy különleges instrukció, ugyanis ennek nincs feladata, hanem ebbe tárolandóak a konstansok. Tehát itt az operand változó nem cím lesz, hanem maga a konstans, amiken a program utasításokat végezhet.

#include <Instruction.h>
A VAR osztály származási diagramja:



Publikus tagfüggvények

- VAR (int operand)
- void executeby (ControlUnit &CU)
- Instruction * clone ()

- ~ VAR ()

3.11.1 Konstruktor és destruktor dokumentációja

VAR::VAR(int operand) [inline]

VAR osztály konstruktora, meghívja az Instrukció osztály konstruktorát az operanddal.

- *Paraméterek:*
  - **operand** -> a tárolandó konstans

VAR::~VAR() [inline]

Destruktor

3.11.2 Tagfüggvények dokumentációja

Instruction *VAR::clone ()

Létrehoz egy dinamikus VAR példányt.

- *Visszatérési érték:*
  - Létrehozott példányra mutató pointer

void executeby(ControlUnit& CU) [inline]

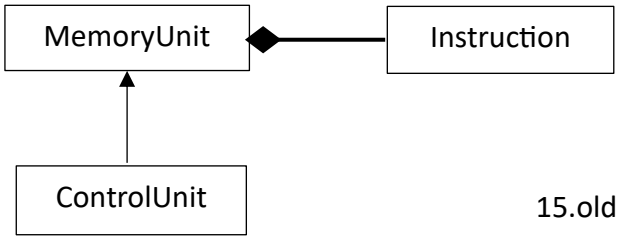
Mivel a VAR osztály konstansokat tárol ezért ennek nincs elvégzendő feladata, így, ha meghívódik ez a függvény akkor dob egy kivételt.

- *Paraméterek:*
  - **CU** -> Vezérlőegység

3.12 **MemoryUnit** osztály

A MemoryUnit osztály a memória megfelelője a Neumann modellben, itt tárolódnak az adatok. Két regisztere van a MAR (Memory Address Register) és a MDR (Memory Data Register).
Az osztályban található egy heterogén kollekció, ami instrukciókat tárol. A tömbben egy szakaszon csak instrukciók utána pedig csak konstansok vannak. Ha a fájl úgy lett megírva, hogy vannak olyan memória címek, amik üresen állnak ott a tömbben null pointer van. Az üres memória címeket a program átlépi és a következő utasítást hajtja végre.

A MemoryUnit osztály származási diagramja:



Privát adattagok:

- int MAR -> Memory Address Register. Ezt tárolja azt a címet, ahonnan ír vagy olvas a memória.
- Instruction* MDR -> Memory Data Register. Olvasáskor ide kerül a kiolvasott instrukció. Íráskor pedig ezt az instrukciót írjuk bele a memóriába.
- Instruction** memory -> Elvégzendő instrukciókat tároló tömb (heterogén kollekció).
- size_t storage -> a memória mérete

Publikus tagfüggvények:

- MemoryUnit (std::string filename)
- int HextoInt (std::string hex)
- void setMAR (int address)
- void setMDR (Instruction ∗mdr)
- Instruction ∗ getMDR ()
- void readEnable ()
- void writeEnable ()
- ∼MemoryUnit ()

3.12.1 Konstruktor és destruktor dokumentációja

MemoryUnit::MemoryUnit (std::string filename)

Konstruktor
Beolvassa az adatokat egy fájlból, majd eltárolja azt egy dinamikusan létrehozott memóriába. A fájl első sora tartalmazza a tömb méretét, amit eltárolunk a storage változóba

- *Paraméterek:*
  - **filename** -> a fájl, ahonnan be kell olvasni az adatokat

MemoryUnit::~MemoryUnit()

Destruktor
Kitörli a dinamikusan foglalt memóriát.

3.12.2 Tagfüggvények dokumentációja

int MemoryUnit::HextoInt(std::string hex)

levágja a nullákat a string elejérõl, majd átváltja integerré. Például 0x0036 karaktersorozatot átalakítja integer 36-tá.

- *Paraméterek:*
  - hex -> az integerré átváltandó string
- *Visszatérési érték:*

- a stringből levágott integer

void MemoryUnit::setMAR(int address) [inline]

Beállítja a MAR-t a megadott értékre

- *Paraméterek:*
  - address -> olvasás vagy írás címe

void MemoryUnit::setMDR(Instruction* mdr) [inline]

Beállítja a MDR-t a megadott értékre

- *Paraméterek:*
  - mdr -> az aktuális memória műveletet (Instrukció)

Instruction* MemoryUnit::getMDR() [inline]

Visszaadjna az MDR értékét

- *Visszatérési érték:*
  - MDR aktuális értéke

void MemoryUnit::readEnable() [inline]

Beolvassa az MDR-be a MAR címen levő instrukciót.

void MemoryUnit::writeEnable() [inline]

Kitöröli a MAR címen lévő adatot a memóriából majd beírja a helyére az MDR-t

size_t MemoryUnit::getStorage() [inline]

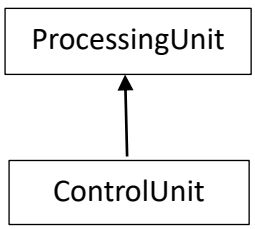
Visszaadjna az storage értékét

- *Visszatérési érték:*
  - storage aktuális értéke

3.13 **ProcessingUnit** osztály

A ProcessingUnit osztály tárolja az éppen betöltött konstanst és segíti az ezen elvégzendő aritmetikai műveleteket.

A ProcessingUnit osztály származási diagramja:



Privát adattagok:

- int ACC -> akkumlátor, tárolja ideiglenesen a számított eredményeket, illetve a betöltött konstansokat

Publikus tagfüggvények:

- void setACC (int acc)
- int getAcc ()
- void add (int value)
- void sub (int value)

3.13.1 Tagfüggvények dokumentációja

void ProcessingUnit::setACC (int acc) [inline]

Beállítja a ACC-t a megadott értékre

- *Paraméterek:*
  - acc -> aktuális konstans

int ProcessingUnit::getAcc ( ) [inline]

Visszaadjna az ACC értékét

- *Visszatérési érték:*
  - ACC aktuális értéke

void ProcessingUnit::add (int value) [inline]

Hozzáadja az ACC-hoz a paraméterként kapott értéket

- *Paraméterek:*
  - value -> hozzáadandó érték

void ProcessingUnit::add (int value) [inline]

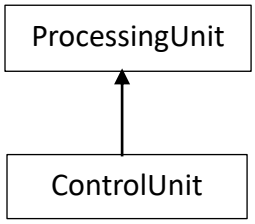
Kivonja az ACC-ból a paraméterként kapott értéket

- *Paraméterek:*
  - value - kivonandó érték

3.14 **IOUnit** osztály

Ez létesíti a kapcsolatot a külvilág és a ”számítógép” között.

A IOUnit osztály származási diagramja:



Privát adattagok:

- std::ostream& os -> Referencia egy kimeneti adatfolyamhoz, amelyre írhatunk adatokat.
- std::istream& is -> Referencia egy bemeneti adatfolyamhoz, amelyről olvashatunk adatokat.

Publikus tagfüggvények:

- IOUnit(std::ostream& os, std::istream& is)
- void print(int var)
- int read()

3.14.1 Konstruktor dokumentációja

IOUnit::IOUnit (std::ostream & os, std::istream & is) [inline]

Konstruktor.

Kiválasztható milyen adatfolyamon akarunk kiírni illet beolvasni.

- *Paraméterek:*
  - os -> kimeneti adatfolyam
  - is -> bemeneti adatfolyam

3.14.2 Tagfüggvények dokumentációja

void IOUnit::print (int var)

Kiírja az os-re, a paraméterként kapott konstanst

- *Paraméterek:*
  - var -> kiírandó konstans

int IOUnit::read ()

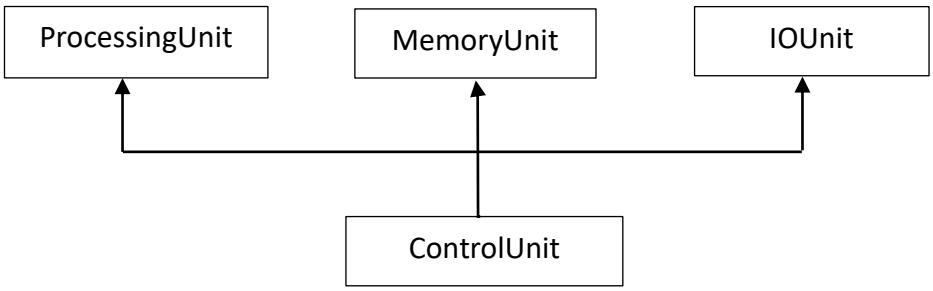
Beolvassa egy egész számot a bemenetről

- *Visszatérési érték:*
  - *a beolvasott szám*

3.15 **ControlUnit** osztály

A ControlUnit Gondoskodik arról, hogy az összes többi ”számítógépes egység” megfelelően és a megfelelő időben hajtsa végre a feladatát.

A ControlUnit osztály származási diagramja:



Privát adattagok:

- int PC -> Programszámláló, megadja a következő instrukció címét
- Instruction* IR -> tárolja az éppen végrehajtandó utasítást

Publikus tagfüggvények:

- ControlUnit(std::string filename, std::ostream& os, std::istream& is)
- void setPC(int val)
- int getPC()
- void cycle()
- Instruction* fetch(int address)

3.15.1 Konstruktor dokumentációja

ControlUnit::ControlUnit(std::string filename, std::ostream& os, std::istream& is) [inline]

Konstruktor.

A paraméterként kapott értékekkel meghívja a MemoryUnit és az IOUnit konstruktorát.

- *Paraméterek:*
  - filename -> beolvasandó fájl
  - os -> Ide ír ki. Default értéke: std::cout.

- is - Innen olvas be. Defaul értéke: std::cin.

3.15.2 Tagfüggvények dokumentációja

void ControlUnit::setPC (int val) [inline]

Beállítja a PC-t a megadott értékre

- *Paraméterek:*
  - PC -> következő instrukció címe

int ControlUnit::getPC ( ) [inline]

Visszaadjna az PC értékét

- *Visszatérési érték:*
  - PC aktuális értéke

void ControlUnit::cycle()

Frissíti az MDR-t a PC alapján a fetch függvény segítségével majd belerakja az IR-be. Ezután megnöveli a PC értékét a következő utasításra majd az IR-ben levő utasítást elvégzi az executeby függvénnyel, ha az nem egy nullpointer.

Instruction* ControlUnit::fetch(int address)

beleteszi a MAR-ba a paraméterként kapott értéket, majd meghívja a MemoryUnit readEnable függvényét. A függvény végül visszaadja az MDR frissített értékét

- *Paraméterek:*
  - address -> következő utasítás címe
- *Visszatérési érték:*
  - az address címen található utasításra mutató pointer

4. Tesztprogram bemutatása:

A tesztprogram a gtest_lite nevű keretrendszert használja a tesztek írásához. Azon kívül előkészített egy txt fájlt, amelyben az n-dik Fibonacci szám kiszámításához szükséges assembly kód található.

A tesztprogramban az alábbi tesztek vannak megvalósítva:

- Beolvasási hiba tesztelése (MemoryUnit, BeolvasHiba): Ellenőrzi, hogy a ControlUnit helyesen dob-e kivételt, ha a fájl megnyitása sikertelen.
- Fájl beolvasása tesztelése (Fibonaccisorozat, fajlbeolvas): Ellenőrzi, hogy a ControlUnit helyesen olvassa be a fájlban található utasításokat és megfelelően tárolja el azokat.

- LOAD utasítás tesztelése (LOAD, executeby): Ellenőrzi, hogy a LOAD utasítás helyesen végrehajtja az operandusában megadott címen található érték betöltését az ACC regiszterbe.
- STORE utasítás tesztelése (STORE, executeby): Ellenőrzi, hogy a STORE utasítás helyesen végrehajtja az ACC regiszter értékének tárolását az operandusában megadott címre.
- ADD utasítás tesztelése (ADD, executeby): Ellenőrzi, hogy az ADD utasítás helyesen végrehajtja az ACC regiszterhez történő hozzáadást az operandusában megadott címen található értékkel.
- SUB utasítás tesztelése (SUB, executeby): Ellenőrzi, hogy a SUB utasítás helyesen végrehajtja az ACC regiszterből történő kivonást az operandusában megadott címen található értékkel.
- JUMP utasítás tesztelése (JUMP, executeby): Ellenőrzi, hogy a JUMP utasítás helyesen végrehajtja a programugrásokat a megfelelő címekre.
- BRANCHGT utasítás tesztelése (BRANCHGT, executeby): Ellenőrzi, hogy a BRANCHGT utasítás helyesen végrehajtja a programugrásokat, ha az ACC regiszter értéke nagyobb, mint az operandusában megadott érték.
- VAR utasítás tesztelése (VAR, executeby): Ellenőrzi, hogy a VAR utasítás helyesen dob-e kivételt, ha próbáljuk végrehajtani.
- READ utasítás tesztelése (READ, executeby): Ellenőrzi, hogy a READ utasítás helyesen olvassa be az adatot a megadott címről.
- PRINT utasítás tesztelése (PRINT, executeby): Ellenőrzi, hogy a PRINT utasítás helyesen írja ki a megadott értéket a kimenetre.

Tehát tesztprogram két fő részből áll. Az első részben a fájlbeolvasást teszteli, majd a második részben minden egyes utasítás osztályának executeby() függvényét ellenőrzi, hogy helyesen működnek-e. Ezen kívül a fetch() és cycle() függvényeket is teszteli, amelyeket a 0., 1., 2. és 9. Fibonacci szám kiszámításával ellenőriz.

A tesztprogramban található egy USER makró, amelynek értékét 1-re lehet állítani, ha a felhasználó szeretné használni a felhasználói felületet.

5. Felhasználói segédlet:

A Program képes egy txt fájlként kapott kódot lefordíteni c++ nyelvre viszont ehhez nagyon fontos úgy megírni a txt fájlt, hogy azt a program képes legyen azt beolvasni.
Először is fontos tudni milyen instrukciók használhatók a szöveges dokumentum megírásához:

LOAD OPERAND -> betölti az OPERAND címen lévő var-t az ACC regiszterbe.

STORE OPERAND -> elmenti az ACC register tartalmát a OPERAND címre var-ként
ADD OPERAND -> hozzáadja az ACC regiszterben lévő értékhez az OPERAND címen lévő értéket.
SUB OPERAND -> kivonja az ACC registerben lévő értékből az OPERAND címen lévő értéket
READ OPERAND -> beolvas az input-ról (std::cin) egy számot és eltárolja az OPERAND címen var-ként
PRINT OPERAND -> kiírja az OPERAND címen lévő var-t az output-ra (std>>cout)
BRANCHGT OPERAND -> a következő utasítás címe OPERAND-ra változik, ha ACC > 0
JUMP OPERAND -> a következő utasítás címe OPERAND-ra változik

Ezek mellett az instrukciók mellett van egy VAR instrukció, ami konstansok tárolására alkalmas.

5.1 A fájl megírásakor a következőket kell figyelembe venni:

- A txt első sora mindig a memória méretét tartalmazza a következő formátumba: 0x0030 ez például azt jelenti, hogy a memória kapacitása 30.
- Az első sor után jönnek az instrukciók címei, és az instrukciók maguk. A következő példa bemutat egy kódot az 1+2 kiszámolására:

ADDR	INSTR	OPERAND	Ez a sor nem kell a txt fájlba
0x0000	LOAD	0x0101	
0x0001	ADD	0x0102	
0x0002	STORE	0x0103	
0x0003	PRINT	0x0103	
0x0101	VAR	0x0001	
0x0102	VAR	0x0002	

- Fontos, hogy a címet és az operandust megfelelő formátumban írjuk, továbbá a VAR konstansokat mindig egy bizonyos cím után helyezzük el, és azok után ne legyenek más típusú utasítások.

Miután elkészült a megfelelő szöveges fájl, amelyet az előzőekben leírtak alapján készítettünk, a USER makrót be kell állítani 1-re. Ezután elindíthatjuk a programot. A program elindítása után a felhasználótól egy fájlnevet kér be a program, amely tartalmazza a lefordított kódot. A program elkezdi végrehajtani a lefordított kódot és megjeleníti az eredményeket vagy végtelen ciklusban fut, amíg a programot le nem állítjuk, vagy el nem ér a VAR instrukciókig.

Források:
https://www.geeksforgeeks.org/computer-organization-basic-computer-instructions/?ref=lbp
http://www.c-jump.com/CIS77/CPU/VonNeumann/lecture.html