**Міністерство освіти І науки України**

**національний університет “Львівська політехніка”**

****

**Кафедра ЕОМ**

лабораторна робота № 6

“ Побудова функціональної схеми процесорів RISC CPU ”

з дисципліни

**"** **Комп’ютерні системи "**

Студента групи КІ-32

Кузишина Назара

Прийняв: Козак Н. Б.

**МЕТА РОБОТИ**

навчитись розробляти нові функціональні модулі з метою розширення функціонування базової архітектури процесорів RISC, що відповідає програмній моделі, а також створення функціональної схеми.

**Завдання**

1. Дослідити програмну модель RISC CPU
2. Визначити склад програмної моделі RISC CPU.
3. Визначити призначення блоків у структурі RISC CPU.
4. Визначити зв’язки між структурними блоками RISC CPU (інтерфейси).
5. Визначити структури інтерфейсів між блоками RISC CPU.
6. Визначити окремі потоки у структурі інтерфейсів:

* інформаційні;

керування

Згідно варіанту в журналі та узгоджено з викладачем обрати один з процесорів заданого сімейства на базі RISC архітектури, в звіті навести основну інформацію по процесору, схему та характеристику.

Внести необхідні зміни в проект з 5-ї лабораторної (типи кешів, розмірності шин даних, назви та типи сигналів, операції, буферні модулі і т.д.) з метою реалізації фукнціоналу з обраного процесору. В звіті представити коротко внесені зміни з описом ходу роботи, а також кінцеву функціональну схему програмної моделі.

З списку вибираємо своє сімейство процесорів, оскільки я 10(2) варіант, то моїм сімейством буде MIPS line

1) ARM сімейство процесорів

2) The MIPS line

3) Hitachi's SuperH

4) Atmel AVR

5) RISC-V, the open source fifth Berkeley RISC ISA.

6) SPARC, by Oracle (previously Sun Microsystems), and Fujitsu.

7) IBM's Power Architecture.

8) Hewlett-Packard's PA-RISC, also known as HP-PA.

**Хід роботи:**

Процесором згідно до якого буде проводитись модифікація моделі RISC CPU був обраний MIPS32 M5150

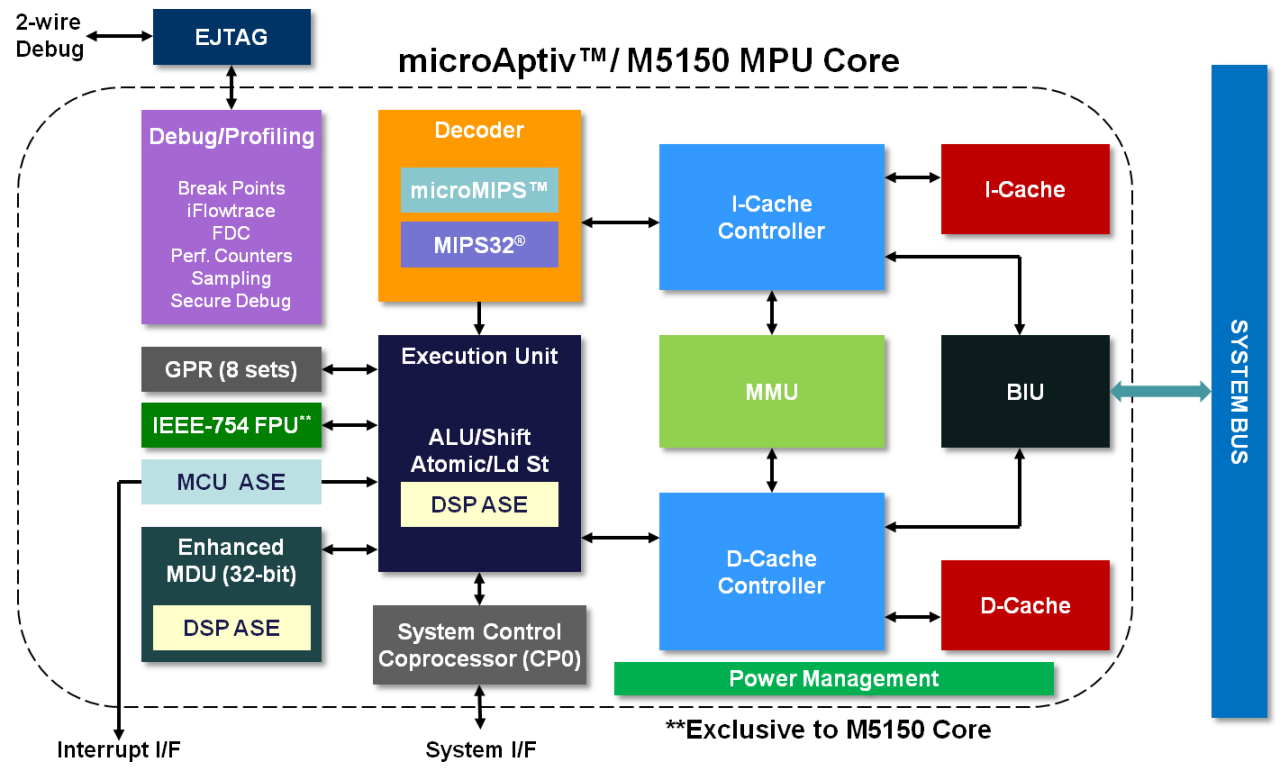
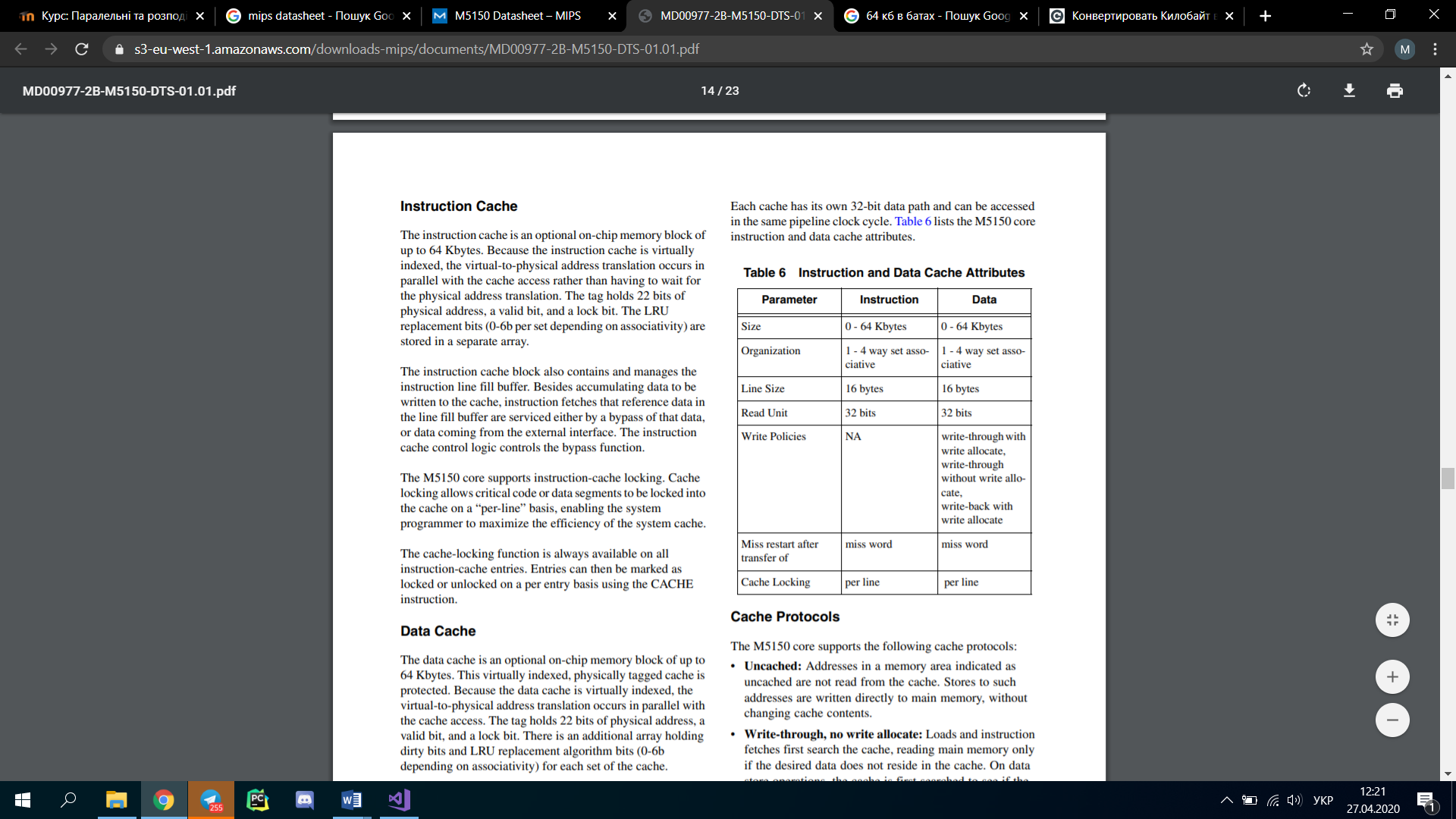


Рис.1. Блок-діаграма процесора MIPS32 M5150

Отже, судячи з даної блок-діаграми, програмна модель RISC CPU не потребує створення додаткових буферних або програмних блоків (оскільки реалізація блоку Power Manager неможлива у програмній моделі), а також видалення вже існуючих. Зміни стосуватимуться лише розрядності шин адрес/даних, а також додавання відповідних команд, яких не вистачає.

**Внесені зміни:**

характеристика буферів процесора MIPS32 M5150



Отже проведу наступні зміни в коді:

*bios.h*

\*\*\*

imemory = new unsigned[65536];

itagmemory = new unsigned[65536];

printf("\*\* ALERT \*\* BIOS: initialize BIOS\n");

for (size = 0; size < 65536; size++) {

imemory[size] = 0xffffffff;

itagmemory[size] = 0xffffffff;

}

Розширюю шину адрес до 32 розрядів, що відповідає функціоналу MIPS-архітектури

#define BOOT\_LENGTH 32

Аналогічно розширив шину адрес (Read Unit) та змінила довжину кодового слова операції в блоці кешу інструкцій згідно функціоналу MIPS-архітектури

*icache.h*

\*\*\*

#define BOOT\_LENGTH 32

#define MAX\_CODE\_LENGTH 800

В файлі *dcache.h* провів аналогічні модифікації довжини вхідного файлу згідно з файломи *bios.h* та *icache.h*

*dcache.h*

*\*\*\**

dmemory = new unsigned[65536];

dsmemory = new unsigned[65536];

dtagmemory = new unsigned[65536];

printf("\*\* ALERT \*\* DCU: initialize Data Cache\n");

\*\*\*

for (i=size; i<65536; i++)

{

dtagmemory[i] = 0xdeadbeef;

dmemory[i] = 0xdeadbeef;

dsmemory[i] = 0;

}}

\*\*\*

Розширюю кількість операцій нашої моделі RISC CPU додаванням до неї операції ділення, що зробить набір виконуваних інструкцій RISC CPU аналогічним до списку інструкцій процесора MIPS32 M5150.

*exec.cpp*

**\*\*\***

case 16: // Операція ділення

dout\_tmp = dina\_tmp / dinb\_tmp;

wait();

break;

**\*\*\***

Розширюю розрядність операндів в блоці MMX згідно до таблиці операцій процесора MIPS32 M5150.

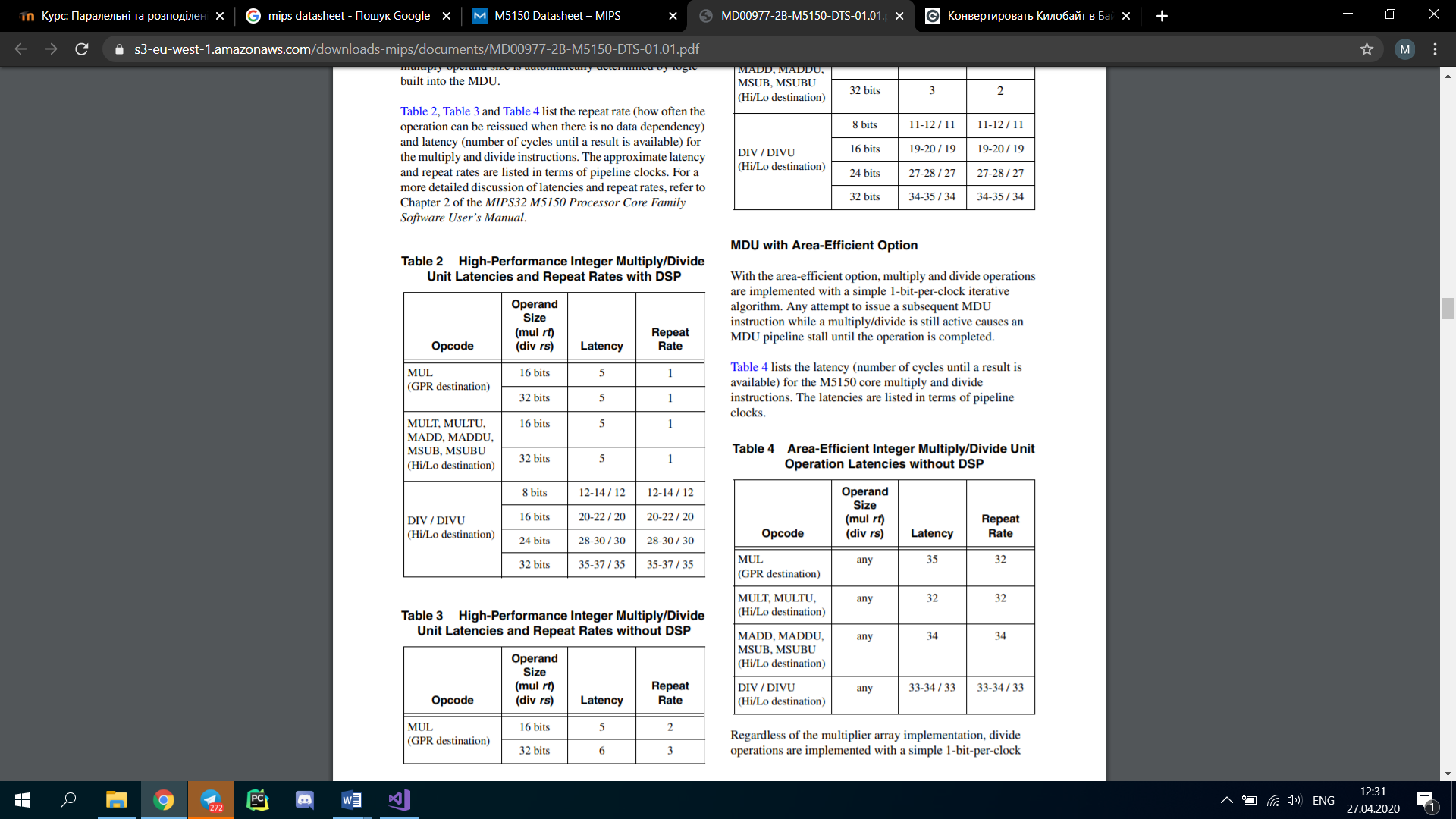


Рис.3. Характеристики та операції блоку обчислень процесора MIPS32 M5150

mmxa0\_tmp = (mmxa\_tmp & 0x000000ff);

mmxb0\_tmp = (mmxb\_tmp & 0x000000ff);

mmxa1\_tmp = (mmxa\_tmp & 0x0000ff00) >> 8;

mmxb1\_tmp = (mmxb\_tmp & 0x0000ff00) >> 8;

mmxa2\_tmp = (mmxa\_tmp & 0x00ff0000) >> 16;

mmxb2\_tmp = (mmxb\_tmp & 0x00ff0000) >> 16;

mmxa3\_tmp = (mmxa\_tmp & 0xff000000) >> 24;

mmxb3\_tmp = (mmxb\_tmp & 0xff000000) >> 24;

mmxa4\_tmp = (mmxa\_tmp & 0xff000000) >> 32;

mmxb4\_tmp = (mmxb\_tmp & 0xff000000) >> 32;

Також збільшую кількість операцій які будуть проводитись над операндами з рухомою комою. До нашого списку команд для моделі RISC CPU додаються дві команди.

1. NFADD – інверсоване додавання над числами з рухомою комою
2. NFSUB – інверсоване віднімання над числами з рухомою комою

*floating.cpp*

*\*\*\**

case 7:

opcode\_encode = "NFADD";

significant\_result = ~(int(fpua\_significand\_tmp) + int(fpub\_significand\_tmp));

wait();

break;

case 8:

opcode\_encode = "NFSUB";

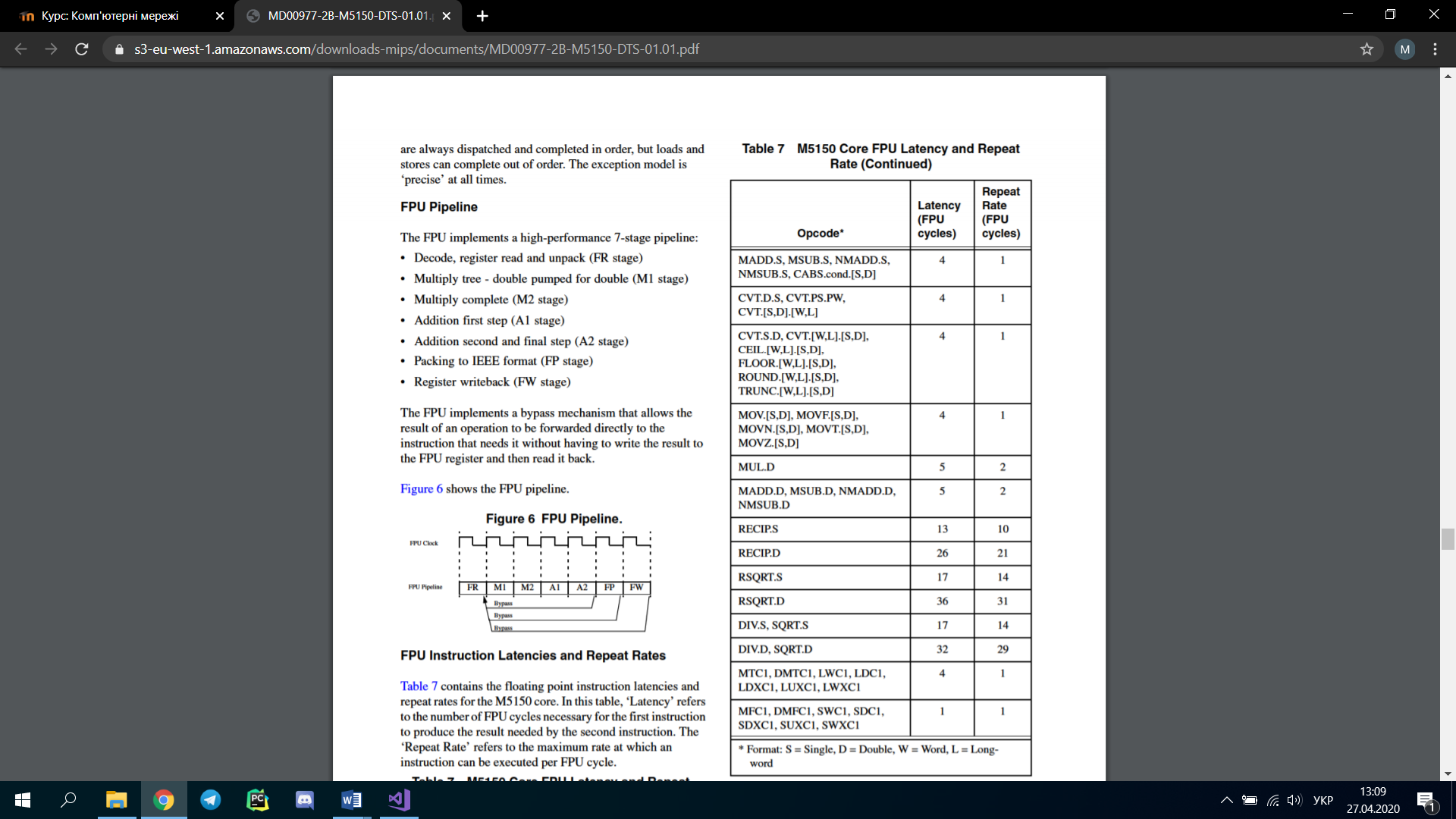
significant\_result = ~(int(fpua\_significand\_tmp) - int(fpub\_significand\_tmp));

wait();

break;

\*\*\*

Усі інші файли даної програмної моделі не потребують модифікації, оскільки повністю збігаються із MIPS-архітектурою, яка є індивідуальним завданням даної лабораторної роботи.



Усі команди цього списку, окрім арифметичних, можуть бути використані лише при написанні програмного коду мовою Assembler, тому їх реалізація у заданій програмній моделі не розглядається

1. **Створення функціональної схеми**



Модифікована функціональна схема за своєю структурою не відрізняється від вихідної програмної моделі, бо в ході роботи даної лабораторної роботи було вказано, що жодні програмні блоки не були видалені або додані. Усі внесені зміни позначені червоним кольором і полягають у зміні розрядності відповідних регістрів та шин адрес даних згідно модифікованої програмної моделі, тобто змін у лістингу програми, що наведені вище.